PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

FD_CLR — macros for synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/select.h>

void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

DESCRIPTION

Refer to *pselect*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

_Exit, _exit — terminate a process

SYNOPSIS

#include <stdlib.h>

void _Exit(int *status*);

#include <unistd.h>

void _exit(int *status*);

DESCRIPTION

For *_Exit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only the least significant 8 bits (that is, *status* & 0377) shall be available from *wait*() and *waitpid*(); the full value shall be available from *waitid*() and in the **siginfo_t** passed to a signal handler for SIGCHLD.

The *_Exit*() and *_exit*() functions shall be functionally equivalent.

The *_Exit*() and *_exit*() functions shall not call functions registered with *atexit*() nor any registered signal handlers. Open streams shall not be flushed. Whether open streams are closed (without flushing) is implementation-defined. Finally, the calling process shall be terminated with the consequences described below.

Consequences of Process Termination

Process termination caused by any reason shall have the following consequences:

Note: These consequences are all extensions to the ISO C standard and are not further CX shaded. However, functionality relating to the XSI option is shaded.

* All of the file descriptors, directory streams, conversion descriptors, and message catalog descriptors open in the calling process shall be closed.

* If the parent process of the calling process has set its SA_NOCLDWAIT flag or has set the action for the SIGCHLD signal to SIG_IGN:

  -- The process' status information (see *Section 2.13*, *Status Information*), if any, shall be discarded.

  -- The lifetime of the calling process shall end immediately. If SA_NOCLDWAIT is set, it is implementation-defined whether a SIGCHLD signal is sent to the parent process.

  -- If a thread in the parent process of the calling process is blocked in *wait*(), *waitpid*(), or *waitid*(), and the parent process has no remaining child processes in the set of waited-for children, the *wait*(), *waitid*(), or *waitpid*() function shall fail and set *errno* to **[ECHILD]**.

  Otherwise:

  -- Status information (see *Section 2.13*, *Status Information*) shall be generated.

  -- The calling process shall be transformed into a zombie process. Its status information shall be made available to the parent process until the process' lifetime ends.

  -- The process' lifetime shall end once its parent obtains the process' status information via a currently-blocked or future call to *wait*(), *waitid*() (without WNOWAIT), or *waitpid*().

  -- If one or more threads in the parent process of the calling process is blocked in a call to *wait*(), *waitid*(), or *waitpid*() awaiting termination of the process, one (or, if any are calling *waitid*() with WNOWAIT, possibly more) of these threads shall obtain the process' status information as specified in *Section 2.13*, *Status Information* and become unblocked.

-- A SIGCHLD shall be sent to the parent process.

* Termination of a process does not directly terminate its children. The sending of a SIGHUP signal as described below indirectly terminates children in some circumstances.

* The parent process ID of all of the existing child processes and zombie processes of the calling process shall be set to the process ID of an implementation-defined system process. That is, these processes shall be inherited by a special system process.

* Each attached shared-memory segment is detached and the value of *shm_nattch* (see *shmget*()) in the data structure associated with its shared memory ID shall be decremented by 1.

* For each semaphore for which the calling process has set a *semadj* value (see *semop*()), that value shall be added to the *semval* of the specified semaphore.

* If the process is a controlling process, the SIGHUP signal shall be sent to each process in the foreground process group of the controlling terminal belonging to the calling process.

* If the process is a controlling process, the controlling terminal associated with the session shall be disassociated from the session, allowing it to be acquired by a new controlling process.

* If the exit of the process causes a process group to become orphaned, and if any member of the newly-orphaned process group is stopped, then a SIGHUP signal followed by a SIGCONT signal shall be sent to each process in the newly-orphaned process group.

* All open named semaphores in the calling process shall be closed as if by appropriate calls to *sem_close*().

* Any memory locks established by the process via calls to *mlockall*() or *mlock*() shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to *_Exit*() or *_exit*().

* Memory mappings that were created in the process shall be unmapped before the process is destroyed.

* Any blocks of typed memory that were mapped in the calling process shall be unmapped, as if *munmap*() was implicitly called to unmap them.

* All open message queue descriptors in the calling process shall be closed as if by appropriate calls to *mq_close*().

* Any outstanding cancelable asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *_Exit*() or *_exit*() operation had not yet occurred, but any associated signal notifications shall be suppressed. The *_Exit*() or *_exit*() operation may block awaiting such I/O completion. Whether any I/O is canceled, and which I/O may be canceled upon *_Exit*() or *_exit*(), is implementation-defined.

* Threads terminated by a call to *_Exit*() or *_exit*() shall not invoke their cancellation cleanup handlers or per-thread data destructors.

* If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described by the *posix_trace_shutdown*() function, and mapping of trace event names to trace event type identifiers of any process built for these trace streams may be deallocated.

RETURN VALUE

These functions do not return.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Normally applications should use *exit*() rather than *_Exit*() or *_exit*().

RATIONALE

Process Termination

Early proposals drew a distinction between normal and abnormal process termination. Abnormal termination was caused only by certain signals and resulted in implementation-defined ''actions'', as discussed below. Subsequent proposals distinguished three types of termination: *normal termination* (as in the current specification), *simple abnormal termination*, and *abnormal termination with actions*. Again the distinction between the two types of abnormal termination was that they were caused by different signals and that implementation-defined actions would result in the latter case. Given that these actions were completely implementation-defined, the early proposals were only saying when the actions could occur and how their occurrence could be detected, but not what they were. This was of little or no use to conforming applications, and thus the distinction is not made in this volume of POSIX.1-2017.

The implementation-defined actions usually include, in most historical implementations, the creation of a file named **core** in the current working directory of the process. This file contains an image of the memory of the process, together with descriptive information about the process, perhaps sufficient to reconstruct the state of the process at the receipt of the signal.

There is a potential security problem in creating a **core** file if the process was set-user-ID and the current user is not the owner of the program, if the process was set-group-ID and none of the user's groups match the group of the program, or if the user does not have permission to write in the current directory. In this situation, an implementation either should not create a **core** file or should make it unreadable by the user.

Despite the silence of this volume of POSIX.1-2017 on this feature, applications are advised not to create files named **core** because of potential conflicts in many implementations. Some implementations use a name other than **core** for the file; for example, by appending the process ID to the filename.

Terminating a Process

It is important that the consequences of process termination as described occur regardless of whether the process called *_exit*() (perhaps indirectly through *exit*()) or instead was terminated due to a signal or for some other reason. Note that in the specific case of *exit*() this means that the *status* argument to *exit*() is treated in the same way as the *status* argument to *_exit*().

A language other than C may have other termination primitives than the C-language *exit*() function, and programs written in such a language should use its native termination primitives, but those should have as part of their function the behavior of *_exit*() as described. Implementations in languages other than C are outside the scope of this version of this volume of POSIX.1-2017, however.

As required by the ISO C standard, using **return** from *main*() has the same behavior (other than with respect to language scope issues) as calling *exit*() with the returned value. Reaching the end of the *main*() function has the same behavior as calling *exit*(0).

A value of zero (or EXIT_SUCCESS, which is required to be zero) for the argument *status* conventionally indicates successful termination. This corresponds to the specification for *exit*() in the ISO C standard. The convention is followed by utilities such as *make* and various shells, which interpret a zero status from a child process as success. For this reason, applications should not call *exit*(0) or *_exit*(0) when they terminate unsuccessfully; for example, in signal-catching functions.

Historically, the implementation-defined process that inherits children whose parents have terminated without waiting on them is called *init* and has a process ID of 1.

The sending of a SIGHUP to the foreground process group when a controlling process terminates corresponds to somewhat different historical implementations. In System V, the kernel sends a SIGHUP on termination of (essentially) a controlling process. In 4.2 BSD, the kernel does not send SIGHUP in a case like this, but the termination of a controlling process is usually noticed by a system daemon, which arranges to send a SIGHUP to the foreground process group with the *vhangup*() function. However, in 4.2 BSD, due to the behavior of the shells that support job control, the controlling process is usually a shell with no other processes in its process group. Thus, a change to make *_exit*() behave this way in such systems should not cause problems with existing applications.

The termination of a process may cause a process group to become orphaned in either of two ways. The connection of a process group to its parent(s) outside of the group depends on both the parents and their children. Thus, a process group may be orphaned by the termination of the last connecting parent process outside of the group or by the termination of the last direct descendant of the parent process(es). In either case, if the termination of a process causes a process group to become orphaned, processes within the group are disconnected from their job control shell, which no longer has any information on the existence of the process group. Stopped processes within the group would languish forever. In order to avoid this problem, newly orphaned process groups that contain stopped processes are sent a SIGHUP signal and a SIGCONT signal to indicate that they have been disconnected from their session. The SIGHUP signal causes the process group members to terminate unless they are catching or ignoring SIGHUP. Under most circumstances, all of the members of the process group are stopped if any of them are stopped.

The action of sending a SIGHUP and a SIGCONT signal to members of a newly orphaned process group is similar to the action of 4.2 BSD, which sends SIGHUP and SIGCONT to each stopped child of an exiting process. If such children exit in response to the SIGHUP, any additional descendants receive similar treatment at that time. In this volume of POSIX.1-2017, the signals are sent to the entire process group at the same time. Also, in this volume of POSIX.1-2017, but not in 4.2 BSD, stopped processes may be orphaned, but may be members of a process group that is not orphaned; therefore, the action taken at *_exit*() must consider processes other than child processes.

It is possible for a process group to be orphaned by a call to *setpgid*() or *setsid*(), as well as by process termination. This volume of POSIX.1-2017 does not require sending SIGHUP and SIGCONT in those cases, because, unlike process termination, those cases are not caused accidentally by applications that are unaware of job control. An implementation can choose to send SIGHUP and SIGCONT in those cases as an extension; such an extension must be documented as required in *<signal.h>*.

The ISO/IEC 9899: 1999 standard adds the *_Exit*() function that results in immediate program termination without triggering signals or *atexit*()-registered functions. In POSIX.1-2008, this is equivalent to the *_exit*() function.

FUTURE DIRECTIONS

None.

SEE ALSO

atexit( ), *exit*( ), *mlock*( ), *mlockall*( ), *mq_close*( ), *munmap*( ), *posix_trace_create*( ), *sem_close*( ), *semop*( ), *setpgid*( ), *setsid*( ), *shmget*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

_longjmp, _setjmp — non-local goto

SYNOPSIS

```
#include <setjmp.h>

void _longjmp(jmp_buf env, int val);
int _setjmp(jmp_buf env);
```

DESCRIPTION

The *_longjmp*() and *_setjmp*() functions shall be equivalent to *longjmp*() and *setjmp*(), respectively, with the additional restriction that *_longjmp*() and *_setjmp*() shall not manipulate the signal mask.

If *_longjmp*() is called even though *env* was never initialized by a call to *_setjmp*(), or when the last such call was in a function that has since returned, the results are undefined.

RETURN VALUE

Refer to *longjmp*( ) and *setjmp*( ).

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If *_longjmp*() is executed and the environment in which *_setjmp*() was executed no longer exists, errors can occur. The conditions under which the environment of the *_setjmp*() no longer exists include exiting the function that contains the *_setjmp*() call, and exiting an inner block with temporary storage. This condition might not be detectable, in which case the *_longjmp*() occurs and, if the environment no longer exists, the contents of the temporary storage of an inner block are unpredictable. This condition might also cause unexpected process termination. If the function has returned, the results are undefined.

Passing *longjmp*() a pointer to a buffer not created by *setjmp*(), passing *_longjmp*() a pointer to a buffer not created by *_setjmp*(), passing *siglongjmp*() a pointer to a buffer not created by *sigsetjmp*(), or passing any of these three functions a buffer that has been modified by the user can cause all the problems listed above, and more.

The *_longjmp*() and *_setjmp*() functions are included to support programs written to historical system interfaces. New applications should use *siglongjmp*() and *sigsetjmp*() respectively.

RATIONALE

None.

FUTURE DIRECTIONS

The *_longjmp*() and *_setjmp*() functions may be removed in a future version.

SEE ALSO

longjmp( ), *setjmp*( ), *siglongjmp*( ), *sigsetjmp*( )

The Base Definitions volume of POSIX.1-2017, **<setjmp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

_tolower — transliterate uppercase characters to lowercase

SYNOPSIS

#include <ctype.h>

int _tolower(int *c*);

DESCRIPTION

The *_tolower*() macro shall be equivalent to *tolower*(*c*) except that the application shall ensure that the argument *c* is an uppercase letter.

RETURN VALUE

Upon successful completion, *_tolower*() shall return the lowercase letter corresponding to the argument passed.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications should use the *tolower*() function instead of the obsolescent *_tolower*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *_tolower*() function may be removed in a future version.

SEE ALSO

tolower( ), *isupper*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer’s Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

_toupper — transliterate lowercase characters to uppercase

SYNOPSIS

```
#include <ctype.h>

int _toupper(int c);
```

DESCRIPTION

The *_toupper*() macro shall be equivalent to *toupper*() except that the application shall ensure that the argument *c* is a lowercase letter.

RETURN VALUE

Upon successful completion, *_toupper*() shall return the uppercase letter corresponding to the argument passed.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications should use the *toupper*() function instead of the obsolescent *_toupper*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *_toupper*() function may be removed in a future version.

SEE ALSO

islower( ), *toupper*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

a64l, l64a — convert between a 32-bit integer and a radix-64 ASCII string

SYNOPSIS

#include <stdlib.h>

long a64l(const char **s*);
char *l64a(long *value*);

DESCRIPTION

These functions maintain numbers stored in radix-64 ASCII characters. This is a notation by which 32-bit integers can be represented by up to six characters; each character represents a digit in radix-64 notation. If the type **long** contains more than 32 bits, only the low-order 32 bits shall be used for these operations.

The characters used to represent digits are **'.'** (dot) for 0, **'/'** for 1, **'0'** through **'9'** for [2,11], **'A'** through **'Z'** for [12,37], and **'a'** through **'z'** for [38,63].

The *a64l*() function shall take a pointer to a radix-64 representation, in which the first digit is the least significant, and return the corresponding **long** value. If the string pointed to by *s* contains more than six characters, *a64l*() shall use the first six. If the first six characters of the string contain a null terminator, *a64l*() shall use only characters preceding the null terminator. The *a64l*() function shall scan the character string from left to right with the least significant digit on the left, decoding each character as a 6-bit radix-64 number. If the type **long** contains more than 32 bits, the resulting value is sign-extended. The behavior of *a64l*() is unspecified if *s* is a null pointer or the string pointed to by *s* was not generated by a previous call to *l64a*().

The *l64a*() function shall take a **long** argument and return a pointer to the corresponding radix-64 representation. The behavior of *l64a*() is unspecified if *value* is negative.

The value returned by *l64a*() may be a pointer into a static buffer. Subsequent calls to *l64a*() may overwrite the buffer.

The *l64a*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *a64l*() shall return the **long** value resulting from conversion of the input string. If a string pointed to by *s* is an empty string, *a64l*() shall return 0L.

The *l64a*() function shall return a pointer to the radix-64 representation. If *value* is 0L, *l64a*() shall return a pointer to an empty string.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the type **long** contains more than 32 bits, the result of *a64l*(*l64a*(*x*)) is *x* in the low-order 32 bits.

RATIONALE

This is not the same encoding as used by either encoding variant of the *uuencode* utility.

FUTURE DIRECTIONS

None.

SEE ALSO

strtoul( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

The Shell and Utilities volume of POSIX.1-2017, *uuencode*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

abort — generate an abnormal process abort

SYNOPSIS

```
#include <stdlib.h>

void abort(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *abort*() function shall cause abnormal process termination to occur, unless the signal SIGABRT is being caught and the signal handler does not return.

The abnormal termination processing shall include the default actions defined for SIGABRT and may include an attempt to effect *fclose*() on all open streams.

The SIGABRT signal shall be sent to the calling process as if by means of *raise*() with the argument SIGABRT.

The status made available to *wait*(), *waitid*(), or *waitpid*() by *abort*() shall be that of a process terminated by the SIGABRT signal. The *abort*() function shall override blocking or ignoring the SIGABRT signal.

RETURN VALUE

The *abort*() function shall not return.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Catching the signal is intended to provide the application developer with a portable means to abort processing, free from possible interference from any implementation-supplied functions.

RATIONALE

The ISO/IEC 9899: 1999 standard requires the *abort*() function to be async-signal-safe. Since POSIX.1-2008 defers to the ISO C standard, this required a change to the DESCRIPTION from "shall include the effect of *fclose*()" to "may include an attempt to effect *fclose*()."

The revised wording permits some backwards-compatibility and avoids a potential deadlock situation.

The Open Group Base Resolution bwg2002-003 is applied, removing the following XSI shaded paragraph from the DESCRIPTION:

"On XSI-conformant systems, in addition the abnormal termination processing shall include the effect of *fclose*() on message catalog descriptors."

There were several reasons to remove this paragraph:

* No special processing of open message catalogs needs to be performed prior to abnormal process termination.
* The main reason to specifically mention that *abort*() includes the effect of *fclose*() on open streams is to flush output queued on the stream. Message catalogs in this context are read-only and, therefore, do not need to be flushed.

- The effect of *fclose*() on a message catalog descriptor is unspecified. Message catalog descriptors are allowed, but not required to be implemented using a file descriptor, but there is no mention in POSIX.1-2008 of a message catalog descriptor using a standard I/O stream FILE object as would be expected by *fclose*().

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *kill*( ), *raise*( ), *signal*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

abs — return an integer absolute value

SYNOPSIS

```
#include <stdlib.h>

int abs(int i);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *abs*() function shall compute the absolute value of its integer operand, *i*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *abs*() function shall return the absolute value of its integer operand.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

In two's-complement representation, the absolute value of the negative integer with largest magnitude {INT_MIN} might not be representable.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fabs( ), *labs*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

accept — accept a new connection on a socket

SYNOPSIS

```
#include <sys/socket.h>

int accept(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

DESCRIPTION

The *accept*() function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket. The file descriptor shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*.

The *accept*() function takes the following arguments:

socket	Specifies a socket that was created with *socket*(), has been bound to an address with *bind*(), and has issued a successful call to *listen*().
address	Either a null pointer, or a pointer to a **sockaddr** structure where the address of the connecting socket shall be returned.
address_len	Either a null pointer, if *address* is a null pointer, or a pointer to a **socklen_t** object which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address.

If *address* is not a null pointer, the address of the peer for the accepted connection shall be stored in the **sockaddr** structure pointed to by *address*, and the length of this address shall be stored in the object pointed to by *address_len*.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

If the listen queue is empty of connection requests and O_NONBLOCK is not set on the file descriptor for the socket, *accept*() shall block until a connection is present. If the *listen*() queue is empty of connection requests and O_NONBLOCK is set on the file descriptor for the socket, *accept*() shall fail and set *errno* to **[EAGAIN]** or **[EWOULDBLOCK]**.

The accepted socket cannot itself accept more connections. The original socket remains open and can accept more connections.

RETURN VALUE

Upon successful completion, *accept*() shall return the non-negative file descriptor of the accepted socket. Otherwise, −1 shall be returned, *errno* shall be set to indicate the error, and any object pointed to by *address_len* shall remain unchanged.

ERRORS

The *accept*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
O_NONBLOCK is set for the socket file descriptor and no connections are present to be accepted.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNABORTED
A connection has been aborted.

EINTR
The *accept*() function was interrupted by a signal that was caught before a valid connection arrived.

EINVAL
The *socket* is not accepting connections.

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum number of file descriptors in the system are already open.

ENOBUFS
No buffer space is available.

ENOMEM
There was insufficient memory available to complete the operation.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The socket type of the specified socket does not support accepting connections.

The *accept*() function may fail if:

EPROTO
A protocol error has occurred; for example, the STREAMS protocol stack has not been initialized.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

When a connection is available, *select*() indicates that the file descriptor for the socket is ready for reading.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *bind*( ), *connect*( ), *listen*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

access, faccessat — determine accessibility of a file descriptor

SYNOPSIS

#include <unistd.h>

int access(const char **path*, int *amode*);

#include <fcntl.h>

int faccessat(int *fd*, const char **path*, int *amode*, int *flag*);

DESCRIPTION

The *access*() function shall check the file named by the pathname pointed to by the *path* argument for accessibility according to the bit pattern contained in *amode*. The checks for accessibility (including directory permissions checked during pathname resolution) shall be performed using the real user ID in place of the effective user ID and the real group ID in place of the effective group ID.

The value of *amode* is either the bitwise-inclusive OR of the access permissions to be checked (R_OK, W_OK, X_OK) or the existence test (F_OK).

If any access permissions are checked, each shall be checked individually, as described in the Base Definitions volume of POSIX.1-2017, *Section 4.5*, *File Access Permissions*, except that where that description refers to execute permission for a process with appropriate privileges, an implementation may indicate success for X_OK even if execute permission is not granted to any user.

The *faccessat*() function, when called with a *flag* value of zero, shall be equivalent to the *access*() function, except in the case where *path* specifies a relative path. In this case the file whose accessibility is to be determined shall be located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *faccessat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and, if flag is zero, the behavior shall be identical to a call to *access*().

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_EACCESS
: The checks for accessibility (including directory permissions checked during pathname resolution) shall be performed using the effective user ID and group ID instead of the real user ID and group ID as required in a call to *access*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EACCES
: Permission bits of the file mode do not permit the requested access, or search permission is denied on a component of the path prefix.

ELOOP
: A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
Write access is requested for a file on a read-only file system.

The *faccessat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

EINVAL
The value of the *amode* argument is invalid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ETXTBSY
Write access is requested for a pure procedure (shared text) file that is being executed.

The *faccessat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

Testing for the Existence of a File

The following example tests whether a file named **myfile** exists in the **/tmp** directory.

```
#include <unistd.h>
...
int result;
const char *pathname = "/tmp/myfile";

result = access (pathname, F_OK);
```

APPLICATION USAGE

Use of these functions is discouraged since by the time the returned information is acted upon, it is out-of-date. (That is, acting upon the information always leads to a time-of-check-to-time-of-use race condition.) An application should instead attempt the action itself and handle the **[EACCES]** error that occurs if the file is not accessible (with a change of effective user and group IDs beforehand, and perhaps a change back afterwards, in the case where *access*() or *faccessat*() without AT_EACCES would have been used.)

Historically, one of the uses of *access*() was in set-user-ID root programs to check whether the user running the program had access to a file. This relied on ``super-user'' privileges which were granted based on the effective user ID being zero, so that when *access*() used the real user ID to check accessibility those privileges were not taken into account. On newer systems where privileges can be assigned which have no association with user or group IDs, if a program with such privileges calls *access*(), the change of IDs has no effect on the privileges and therefore they are taken into account in the accessibility checks. Thus, *access*() (and *faccessat*() with flag zero) cannot be used for this historical purpose in such programs. Likewise, if a system provides any additional or alternate file access control mechanisms that are not user ID-based, they will still be taken into account.

If a relative pathname is used, no account is taken of whether the current directory (or the directory associated with the file descriptor *fd*) is accessible via any absolute pathname. Applications using *access*(), or *faccessat*() without AT_EACCES, may consequently act as if the file would be accessible to a user with the real user ID and group ID of the process when such a user would not in practice be able to access the file because access would be denied at some point above the current directory (or the directory associated with the file descriptor *fd*) in the file hierarchy.

If *access*() or *faccessat*() is used with W_OK to check for write access to a directory which has the S_ISVTX bit set, a return value indicating the directory is writable can be misleading since some operations on files in the directory would not be permitted based on the ownership of those files (see the Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*).

Additional values of *amode* other than the set defined in the description may be valid; for example, if a system has extended access controls.

The use of the AT_EACCESS value for *flag* enables functionality not available in *access*().

RATIONALE

In early proposals, some inadequacies in the *access*() function led to the creation of an *eaccess*() function because:

1. Historical implementations of *access*() do not test file access correctly when the process' real user ID is superuser. In particular, they always return zero when testing execute permissions without regard to whether the file is executable.
2. The superuser has complete access to all files on a system. As a consequence, programs started by the superuser and switched to the effective user ID with lesser privileges cannot use *access*() to test their file access permissions.

However, the historical model of *eaccess*() does not resolve problem (1), so this volume of POSIX.1-2017 now allows *access*() to behave in the desired way because several implementations have corrected the problem. It was also argued that problem (2) is more easily solved by using *open*(), *chdir*(), or one of the *exec* functions as appropriate and responding to the error, rather than creating a new function that would not be as reliable. Therefore, *eaccess*() is not included in this volume of POSIX.1-2017.

The sentence concerning appropriate privileges and execute permission bits reflects the two possibilities implemented by historical implementations when checking superuser access for X_OK.

New implementations are discouraged from returning X_OK unless at least one execution permission bit is set.

The purpose of the *faccessat*() function is to enable the checking of the accessibility of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *access*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *faccessat*() function it can be guaranteed that the file tested for accessibility is located relative to the desired directory.

FUTURE DIRECTIONS

These functions may be formally deprecated (for example, by shading them OB) in a future version of this standard.

SEE ALSO

chmod( ), *fstatat*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.5*, *File Access Permissions*, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

acos, acosf, acosl — arc cosine functions

SYNOPSIS

```
#include <math.h>

double acos(double x);
float acosf(float x);
long double acosl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the principal value of the arc cosine of their argument *x*. The value of *x* should be in the range [−1,1].

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc cosine of *x*, in the range $[0,\pi]$ radians.

For finite values of *x* not in the range [−1,1], a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is +1, +0 shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is finite and is not in the range [−1,1], or is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cos( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

acosh, acoshf, acoshl — inverse hyperbolic cosine functions

SYNOPSIS

```
#include <math.h>

double acosh(double x);
float acoshf(float x);
long double acoshl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic cosine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic cosine of their argument.

For finite values of $x < 1$, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is +1, +0 shall be returned.

If *x* is +Inf, +Inf shall be returned.

If *x* is −Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is finite and less than +1.0, or is −Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cosh( ), *feclearexcept*( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

acosl — arc cosine functions

SYNOPSIS

#include <math.h>

long double acosl(long double *x*);

DESCRIPTION

Refer to *acos*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_cancel — cancel an asynchronous I/O request

SYNOPSIS

```
#include <aio.h>

int aio_cancel(int fildes, struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_cancel*() function shall attempt to cancel one or more asynchronous I/O requests currently outstanding against file descriptor *fildes*. The *aiocbp* argument points to the asynchronous I/O control block for a particular request to be canceled. If *aiocbp* is NULL, then all outstanding cancelable asynchronous I/O requests against *fildes* shall be canceled.

Normal asynchronous notification shall occur for asynchronous I/O operations that are successfully canceled. If there are requests that cannot be canceled, then the normal asynchronous completion process shall take place for those requests when they are completed.

For requested operations that are successfully canceled, the associated error status shall be set to **[ECANCELED]** and the return status shall be −1. For requested operations that are not successfully canceled, the *aiocbp* shall not be modified by *aio_cancel*().

If *aiocbp* is not NULL, then if *fildes* does not have the same value as the file descriptor with which the asynchronous operation was initiated, unspecified results occur.

Which operations are cancelable is implementation-defined.

RETURN VALUE

The *aio_cancel*() function shall return the value AIO_CANCELED if the requested operation(s) were canceled. The value AIO_NOTCANCELED shall be returned if at least one of the requested operation(s) cannot be canceled because it is in progress. In this case, the state of the other operations, if any, referenced in the call to *aio_cancel*() is not indicated by the return value of *aio_cancel*(). The application may determine the state of affairs for these operations by using *aio_error*(). The value AIO_ALLDONE is returned if all of the operations have already completed. Otherwise, the function shall return −1 and set *errno* to indicate the error.

ERRORS

The *aio_cancel*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read( ), *aio_write*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_error — retrieve errors status for an asynchronous I/O operation

SYNOPSIS

```
#include <aio.h>

int aio_error(const struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_error*() function shall return the error status associated with the **aiocb** structure referenced by the *aiocbp* argument. The error status for an asynchronous I/O operation is the *errno* value that would be set by the corresponding *read*(), *write*(), *fdatasync*(), or *fsync*() operation. If the operation has not yet completed, then the error status shall be equal to **[EINPROGRESS]**.

If the **aiocb** structure pointed to by *aiocbp* is not associated with an operation that has been scheduled, the results are undefined.

RETURN VALUE

If the asynchronous I/O operation has completed successfully, then 0 shall be returned. If the asynchronous operation has completed unsuccessfully, then the error status, as described for *read*(), *write*(), *fdatasync*(), and *fsync*(), shall be returned. If the asynchronous I/O operation has not yet completed, then **[EINPROGRESS]** shall be returned.

If the *aio_error*() function fails, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *aio_error*() function may fail if:

EINVAL

The *aiocbp* argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_fsync*( ), *aio_read*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *read*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_fsync — asynchronous file synchronization

SYNOPSIS

#include <aio.h>

int aio_fsync(int *op*, struct aiocb **aiocbp*);

DESCRIPTION

The *aio_fsync*() function shall asynchronously perform a file synchronization operation, as specified by the *op* argument, for I/O operations associated with the file indicated by the file descriptor *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument and queued at the time of the call to *aio_fsync*(). The function call shall return when the synchronization request has been initiated or queued to the file or device (even when the data cannot be synchronized immediately).

If *op* is O_DSYNC, all currently queued I/O operations shall be completed as if by a call to *fdatasync*(); that is, as defined for synchronized I/O data integrity completion.

If *op* is O_SYNC, all currently queued I/O operations shall be completed as if by a call to *fsync*(); that is, as defined for synchronized I/O file integrity completion. If the *aio_fsync*() function fails, or if the operation queued by *aio_fsync*() fails, then outstanding I/O operations are not guaranteed to have been completed.

If *aio_fsync*() succeeds, then it is only the I/O that was queued at the time of the call to *aio_fsync*() that is guaranteed to be forced to the relevant completion state. The completion of subsequent I/O on the file descriptor is not guaranteed to be completed in a synchronized fashion.

The *aiocbp* argument refers to an asynchronous I/O control block. The *aiocbp* value may be used as an argument to *aio_error*() and *aio_return*() in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. When the request is queued, the error status for the operation is **[EINPROGRESS]**. When all data has been successfully transferred, the error status shall be reset to reflect the success or failure of the operation. If the operation does not complete successfully, the error status for the operation shall be set to indicate the error. The *aio_sigevent* member determines the asynchronous notification to occur as specified in *Section 2.4.1*, *Signal Generation and Delivery* when all operations have achieved synchronized I/O completion. All other members of the structure referenced by *aiocbp* are ignored. If the control block referenced by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If the *aio_fsync*() function fails or *aiocbp* indicates an error condition, data is not guaranteed to have been successfully transferred.

RETURN VALUE

The *aio_fsync*() function shall return the value 0 if the I/O operation is successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

ERRORS

The *aio_fsync*() function shall fail if:

EAGAIN
The requested asynchronous operation was not queued due to temporary resource limitations.

EBADF
The *aio_fildes* member of the **aiocb** structure referenced by the *aiocbp* argument is not a valid file descriptor.

EINVAL
This implementation does not support synchronized I/O for this file.

EINVAL
The *aio_fildes* member of the **aiocb** structure refers to a file on which an *fsync*() operation is not possible.

EINVAL
A value of *op* other than O_DSYNC or O_SYNC was specified, or O_DSYNC was specified and the implementation does not provide runtime support for the Synchronized Input and Output option, or O_SYNC was specified and the implementation does not provide runtime support for the File Synchronization option.

In the event that any of the queued I/O operations fail, *aio_fsync*() shall return the error condition defined for *read*() and *write*(). The error is returned in the error status for the asynchronous operation, which can be retrieved using *aio_error*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Note that even if the file descriptor is not open for writing, if there are any pending write requests on the underlying file, then that I/O will be completed prior to the return of a call to *aio_error*() or *aio_return*() indicating that the operation has completed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_error( ), *aio_return*( ), *fcntl*( ), *fdatasync*( ), *fsync*( ), *open*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_read — asynchronous read from a file

SYNOPSIS

```
#include <aio.h>

int aio_read(struct aiocb *aiocbp);
```

DESCRIPTION

The *aio_read*() function shall read *aiocbp*−>*aio_nbytes* from the file associated with *aiocbp*−>*aio_fildes* into the buffer pointed to by *aiocbp*−>*aio_buf*. The function call shall return when the read request has been initiated or queued to the file or device (even when the data cannot be delivered immediately).

If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to a base scheduling priority minus *aiocbp*−>*aio_reqprio*. If Thread Execution Scheduling is not supported, then the base scheduling priority is that of the calling process;
otherwise, the base scheduling priority is that of the calling thread.

The *aiocbp* value may be used as an argument to *aio_error*() and *aio_return*() in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding. If an error condition is encountered during queuing, the function call shall return without having initiated or queued the request. The requested operation takes place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aio_sigevent* member specifies the notification which occurs when the request is completed.

The *aiocbp*−>*aio_lio_opcode* field shall be ignored by *aio_read*().

The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp*−>*aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

If synchronized I/O is enabled on the file associated with *aiocbp*−>*aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp*−>*aio_fildes*.

RETURN VALUE

The *aio_read*() function shall return the value zero if the I/O operation is successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

ERRORS

The *aio_read*() function shall fail if:

EAGAIN
The requested asynchronous I/O operation was not queued due to system resource limitations.

Each of the following conditions may be detected synchronously at the time of the call to *aio_read*(), or asynchronously. If any of the conditions below are detected synchronously, the *aio_read*() function shall return −1 and set *errno* to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation is set to −1, and the error status of the asynchronous operation is set to the corresponding value.

EBADF
The *aiocbp*−>*aio_fildes* argument is not a valid file descriptor open for reading.

EINVAL
The file offset value implied by *aiocbp*−>*aio_offset* would be invalid, *aiocbp*−>*aio_reqprio* is not a valid value, or *aiocbp*−>*aio_nbytes* is an invalid value.

In the case that the *aio_read*() successfully queues the I/O operation but the operation is subsequently canceled or encounters an error, the return status of the asynchronous operation is one of the values normally returned by the *read*() function call. In addition, the error status of the asynchronous operation is set to one of the error statuses normally set by the *read*() function call, or one of the following values:

EBADF
The *aiocbp*−>*aio_fildes* argument is not a valid file descriptor open for reading.

ECANCELED
The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*() request.

EINVAL
The file offset value implied by *aiocbp*−>*aio_offset* would be invalid.

The following condition may be detected synchronously or asynchronously:

EOVERFLOW
The file is a regular file, *aiobcp*−>*aio_nbytes* is greater than 0, and the starting offset in *aiobcp*−>*aio_offset* is before the end-of-file and is at or beyond the offset maximum in the open file description associated with *aiocbp*−>*aio_fildes*.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
aio_cancel( ), *aio_error*( ), *lio_listio*( ), *aio_return*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ), *read*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_return — retrieve return status of an asynchronous I/O operation

SYNOPSIS

#include <aio.h>

ssize_t aio_return(struct aiocb **aiocbp*);

DESCRIPTION

The *aio_return*() function shall return the return status associated with the **aiocb** structure referenced by the *aiocbp* argument. The return status for an asynchronous I/O operation is the value that would be returned by the corresponding *read*(), *write*(), or *fsync*() function call. If the error status for the operation is equal to **[EINPROGRESS]**, then the return status for the operation is undefined. The *aio_return*() function may be called exactly once to retrieve the return status of a given asynchronous operation; thereafter, if the same **aiocb** structure is used in a call to *aio_return*() or *aio_error*(), an error may be returned. When the **aiocb** structure referred to by *aiocbp* is used to submit another asynchronous operation, then *aio_return*() may be successfully used to retrieve the return status of that operation.

RETURN VALUE

If the asynchronous I/O operation has completed, then the return status, as described for *read*(), *write*(), and *fsync*(), shall be returned. If the asynchronous I/O operation has not yet completed, the results of *aio_return*() are undefined.

If the *aio_return*() function fails, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *aio_return*() function may fail if:

EINVAL

The *aiocbp* argument does not refer to an asynchronous operation whose return status has not yet been retrieved.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_cancel( ), *aio_error*( ), *aio_fsync*( ), *aio_read*( ), *aio_write*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *read*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_suspend — wait for an asynchronous I/O request

SYNOPSIS

```
#include <aio.h>

int aio_suspend(const struct aiocb *const list[], int nent,
   const struct timespec *timeout);
```

DESCRIPTION

The *aio_suspend*() function shall suspend the calling thread until at least one of the asynchronous I/O operations referenced by the *list* argument has completed, until a signal interrupts the function, or, if *timeout* is not NULL, until the time interval specified by *timeout* has passed. If any of the **aiocb** structures in the list correspond to completed asynchronous I/O operations (that is, the error status for the operation is not equal to **[EINPROGRESS]**) at the time of the call, the function shall return without suspending the calling thread. The *list* argument is an array of pointers to asynchronous I/O control blocks. The *nent* argument indicates the number of elements in the array. Each **aiocb** structure pointed to has been used in initiating an asynchronous I/O request via *aio_read*(), *aio_write*(), or *lio_listio*(). This array may contain null pointers, which are ignored. If this array contains pointers that refer to **aiocb** structures that have not been used in submitting asynchronous I/O, the effect is undefined.

If the time interval indicated in the **timespec** structure pointed to by *timeout* passes before any of the I/O operations referenced by *list* are completed, then *aio_suspend*() shall return with an error. If the Monotonic Clock option is supported, the clock that shall be used to measure this time interval shall be the CLOCK_MONOTONIC clock.

RETURN VALUE

If the *aio_suspend*() function returns after one or more asynchronous I/O operations have completed, the function shall return zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

The application may determine which asynchronous I/O completed by scanning the associated error and return status using *aio_error*() and *aio_return*(), respectively.

ERRORS

The *aio_suspend*() function shall fail if:

EAGAIN
No asynchronous I/O indicated in the list referenced by *list* completed in the time interval indicated by *timeout*.

EINTR
A signal interrupted the *aio_suspend*() function. Note that, since each asynchronous I/O operation may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read( ), *aio_write*( ), *lio_listio*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

aio_write — asynchronous write to a file

SYNOPSIS

#include <aio.h>

int aio_write(struct aiocb **aiocbp*);

DESCRIPTION

The *aio_write*() function shall write *aiocbp*−>*aio_nbytes* to the file associated with *aiocbp*−>*aio_fildes* from the buffer pointed to by *aiocbp*−>*aio_buf*. The function shall return when the write request has been initiated or, at a minimum, queued to the file or device.

If prioritized I/O is supported for this file, then the asynchronous operation shall be submitted at a priority equal to a base scheduling priority minus *aiocbp*−>*aio_reqprio*. If Thread Execution Scheduling is not supported, then the base scheduling priority is that of the calling process;
otherwise, the base scheduling priority is that of the calling thread.

The *aiocbp* argument may be used as an argument to *aio_error*() and *aio_return*() in order to determine the error status and return status, respectively, of the asynchronous operation while it is proceeding.

The *aiocbp* argument points to an **aiocb** structure. If the buffer pointed to by *aiocbp*−>*aio_buf* or the control block pointed to by *aiocbp* becomes an illegal address prior to asynchronous I/O completion, then the behavior is undefined.

If O_APPEND is not set for the file descriptor *aio_fildes*, then the requested operation shall take place at the absolute position in the file as given by *aio_offset*, as if *lseek*() were called immediately prior to the operation with an *offset* equal to *aio_offset* and a *whence* equal to SEEK_SET. If O_APPEND is set for the file descriptor, or if *aio_fildes* is associated with a device that is incapable of seeking, write operations append to the file in the same order as the calls were made, except under circumstances described in *Section 2.8.2*, *Asynchronous I/O*. After a successful call to enqueue an asynchronous I/O operation, the value of the file offset for the file is unspecified.

The *aio_sigevent* member specifies the notification which occurs when the request is completed.

The *aiocbp*−>*aio_lio_opcode* field shall be ignored by *aio_write*().

Simultaneous asynchronous operations using the same *aiocbp* produce undefined results.

If synchronized I/O is enabled on the file associated with *aiocbp*−>*aio_fildes*, the behavior of this function shall be according to the definitions of synchronized I/O data integrity completion, and synchronized I/O file integrity completion.

For any system action that changes the process memory space while an asynchronous I/O is outstanding to the address range being changed, the result of that action is undefined.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp*−>*aio_fildes*.

RETURN VALUE

The *aio_write*() function shall return the value zero if the I/O operation is successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

ERRORS

The *aio_write*() function shall fail if:

EAGAIN
The requested asynchronous I/O operation was not queued due to system resource limitations.

Each of the following conditions may be detected synchronously at the time of the call to *aio_write*(), or asynchronously. If any of the conditions below are detected synchronously, the *aio_write*() function shall return −1 and set *errno* to the corresponding value. If any of the conditions below are detected asynchronously, the return status of the asynchronous operation shall be set to −1, and the error status of the asynchronous operation is set to the corresponding value.

EBADF
The *aiocbp*−>*aio_fildes* argument is not a valid file descriptor open for writing.

EINVAL
The file offset value implied by *aiocbp*−>*aio_offset* would be invalid,
aiocbp−>*aio_reqprio* is not a valid value, or *aiocbp*−>*aio_nbytes* is an invalid value.

In the case that the *aio_write*() successfully queues the I/O operation, the return status of the asynchronous operation shall be one of the values normally returned by the *write*() function call. If the operation is successfully queued but is subsequently canceled or encounters an error, the error status for the asynchronous operation contains one of the values normally set by the *write*() function call, or one of the following:

EBADF
The *aiocbp*−>*aio_fildes* argument is not a valid file descriptor open for writing.

EINVAL
The file offset value implied by *aiocbp*−>*aio_offset* would be invalid.

ECANCELED
The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*() request.

The following condition may be detected synchronously or asynchronously:

EFBIG
The file is a regular file, *aiobcp*−>*aio_nbytes* is greater than 0, and the starting offset in *aiobcp*−>*aio_offset* is at or beyond the offset maximum in the open file description associated with *aiocbp*−>*aio_fildes*.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Section 2.8.2, *Asynchronous I/O*, *aio_cancel*( ), *aio_error*( ), *aio_read*( ), *aio_return*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lio_listio*( ), *lseek*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

alarm — schedule an alarm signal

SYNOPSIS

#include <unistd.h>

unsigned alarm(unsigned *seconds*);

DESCRIPTION

The *alarm*() function shall cause the system to generate a SIGALRM signal for the process after the number of realtime seconds specified by *seconds* have elapsed. Processor scheduling delays may prevent the process from handling the signal as soon as it is generated.

If *seconds* is 0, a pending alarm request, if any, is canceled.

Alarm requests are not stacked; only one SIGALRM generation can be scheduled in this manner. If the SIGALRM signal has not yet been generated, the call shall result in rescheduling the time at which the SIGALRM signal is generated.

Interactions between *alarm*() and *setitimer*() are unspecified.

RETURN VALUE

If there is a previous *alarm*() request with time remaining, *alarm*() shall return a non-zero value that is the number of seconds until the previous request would have generated a SIGALRM signal. Otherwise, *alarm*() shall return 0.

ERRORS

The *alarm*() function is always successful, and no return value is reserved to indicate an error.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *fork*() function clears pending alarms in the child process. A new process image created by one of the *exec* functions inherits the time left to an alarm signal in the image of the old process.

Application developers should note that the type of the argument *seconds* and the return value of *alarm*() is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Application developers should be aware of possible interactions when the same process uses both the *alarm*() and *sleep*() functions.

RATIONALE

Many historical implementations (including Version 7 and System V) allow an alarm to occur up to a second early. Other implementations allow alarms up to half a second or one clock tick early or do not allow them to occur early at all. The latter is considered most appropriate, since it gives the most predictable behavior, especially since the signal can always be delayed for an indefinite amount of time due to scheduling. Applications can thus choose the *seconds* argument as the minimum amount of time they wish to have elapse before the signal.

The term "realtime" here and elsewhere (*sleep*(), *times*()) is intended to mean "wall clock" time as common English usage, and has nothing to do with "realtime operating systems". It is in contrast to *virtual time*, which could be misinterpreted if just *time* were used.

In some implementations, including 4.3 BSD, very large values of the *seconds* argument are silently rounded down to an implementation-specific maximum value. This maximum is large enough (to the order of several months) that the effect is not noticeable.

There were two possible choices for alarm generation in multi-threaded applications: generation for the calling thread or generation for the process. The first option would not have been particularly useful since the alarm state is maintained on a per-process basis and the alarm that is established by the last invocation of *alarm*() is the only one that would be active.

Furthermore, allowing generation of an asynchronous signal for a thread would have introduced an exception to the overall signal model. This requires a compelling reason in order to be justified.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fork*( ), *getitimer*( ), *pause*( ), *sigaction*( ), *sleep*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

alphasort, scandir — scan a directory

SYNOPSIS

```
#include <dirent.h>

int alphasort(const struct dirent **d1, const struct dirent **d2);
int scandir(const char *dir, struct dirent ***namelist,
   int (*sel)(const struct dirent *),
   int (*compar)(const struct dirent **, const struct dirent **));
```

DESCRIPTION

The *alphasort*() function can be used as the comparison function for the *scandir*() function to sort the directory entries, *d1* and *d2*, into alphabetical order. Sorting happens as if by calling the *strcoll*() function on the *d_name* element of the **dirent** structures passed as the two parameters. If the *strcoll*() function fails, the return value of *alphasort*() is unspecified.

The *alphasort*() function shall not change the setting of *errno* if successful. Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *alphasort*(), then check *errno*.

The *scandir*() function shall scan the directory *dir*, calling the function referenced by *sel* on each directory entry. Entries for which the function referenced by *sel* returns non-zero shall be stored in strings allocated as if by a call to *malloc*(), and sorted as if by a call to *qsort*() with the comparison function *compar*, except that *compar* need not provide total ordering. The strings are collected in array *namelist* which shall be allocated as if by a call to *malloc*(). If *sel* is a null pointer, all entries shall be selected. If the comparison function *compar* does not provide total ordering, the order in which the directory entries are stored is unspecified.

RETURN VALUE

Upon successful completion, the *alphasort*() function shall return an integer greater than, equal to, or less than 0, according to whether the name of the directory entry pointed to by *d1* is lexically greater than, equal to, or less than the directory pointed to by *d2* when both are interpreted as appropriate to the current locale. There is no return value reserved to indicate an error.

Upon successful completion, the *scandir*() function shall return the number of entries in the array and a pointer to the array through the parameter *namelist*. Otherwise, the *scandir*() function shall return −1.

ERRORS

The *scandir*() function shall fail if:

EACCES
: Search permission is denied for the component of the path prefix of *dir* or read permission is denied for *dir*.

ELOOP
: A loop exists in symbolic links encountered during resolution of the *dir* argument.

ENAMETOOLONG
: The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
: A component of *dir* does not name an existing directory or *dir* is an empty string.

ENOMEM
: Insufficient storage space is available.

ENOTDIR
A component of *dir* names an existing file that is neither a directory nor a symbolic link to a directory.

EOVERFLOW
One of the values to be returned or passed to a callback function cannot be represented correctly.

The *scandir*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dir* argument.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE
Too many files are currently open in the system.

The following sections are informative.

EXAMPLES

An example to print the files in the current directory:

```
#include <dirent.h>
#include <stdio.h>
#include <stdlib.h>
...
struct dirent **namelist;
int i,n;

   n = scandir(".", &namelist, 0, alphasort);
   if (n < 0)
      perror("scandir");
   else {
      for (i = 0; i < n; i++) {
         printf("%s\n", namelist[i]->d_name);
         free(namelist[i]);
         }
      }
   free(namelist);
...
```

APPLICATION USAGE

If *dir* contains filenames that do not form character strings, or which contain characters outside the domain of the collating sequence of the current locale, the *alphasort*() function need not provide a total ordering. This condition is not possible if all filenames within the directory consist only of characters from the portable filename character set.

The *scandir*() function may allocate dynamic storage during its operation. If *scandir*() is forcibly terminated, such as by *longjmp*() or *siglongjmp*() being executed by the function pointed to by *sel* or *compar*, or by an interrupt routine, *scandir*() does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, then wait until *scandir*() returns to act on the interrupt.

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *scandir*(), this is *namelist* (including all of the individual strings in *namelist*).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

qsort( ), *strcoll*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

asctime, asctime_r — convert date and time to a string

SYNOPSIS

```
#include <time.h>

char *asctime(const struct tm *timeptr);
char *asctime_r(const struct tm *restrict tm, char *restrict buf);
```

DESCRIPTION

For *asctime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *asctime*() function shall convert the broken-down time in the structure pointed to by *timeptr* into a string in the form:

```
Sun Sep 16 01:03:52 1973\n\0
```

using the equivalent of the following algorithm:

```
char *asctime(const struct tm *timeptr)
{
    static char wday_name[7][3] = {
        "Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"
    };
    static char mon_name[12][3] = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun",
        "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
    };
    static char result[26];

    sprintf(result, "%.3s %.3s%3d %.2d:%.2d:%.2d %d\n",
        wday_name[timeptr->tm_wday],
        mon_name[timeptr->tm_mon],
        timeptr->tm_mday, timeptr->tm_hour,
        timeptr->tm_min, timeptr->tm_sec,
        1900 + timeptr->tm_year);
    return result;
}
```

However, the behavior is undefined if *timeptr−>tm_wday* or *timeptr−>tm_mon* are not within the normal ranges as defined in *<time.h>*, or if *timeptr−>tm_year* exceeds {INT_MAX}−1990, or if the above algorithm would attempt to generate more than 26 bytes of output (including the terminating null).

The **tm** structure is defined in the *<time.h>* header.

The *asctime*(), *ctime*(), *gmtime*(), and *localtime*() functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *asctime*() function need not be thread-safe.

The *asctime_r*() function shall convert the broken-down time in the structure pointed to by *tm* into a string (of the same form as that returned by *asctime*(), and with the same undefined behavior when input or output is out of range) that is placed in the user-supplied buffer pointed to by *buf* (which shall contain at least 26 bytes) and then return *buf*.

RETURN VALUE

Upon successful completion, *asctime*() shall return a pointer to the string. If the function is unsuccessful, it shall return NULL.

Upon successful completion, *asctime_r*() shall return a pointer to a character string containing the date and time. This string is pointed to by the argument *buf*. If the function is unsuccessful, it shall return NULL.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are included only for compatibility with older implementations. They have undefined behavior if the resulting string would be too long, so the use of these functions should be discouraged. On implementations that do not detect output string length overflow, it is possible to overflow the output buffers in such a way as to cause applications to fail, or possible system security violations. Also, these functions do not support localized date and time formats. To avoid these problems, applications should use *strftime*() to generate strings from broken-down times.

Values for the broken-down time structure can be obtained by calling *gmtime*() or *localtime*().

The *asctime_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The standard developers decided to mark the *asctime*() and *asctime_r*() functions obsolescent even though *asctime*() is in the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides the *strftime*() function which can be used to avoid these problems.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

clock( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

asin, asinf, asinl — arc sine function

SYNOPSIS

```
#include <math.h>

double asin(double x);
float asinf(float x);
long double asinl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the principal value of the arc sine of their argument *x*. The value of *x* should be in the range [−1,1].

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc sine of *x*, in the range [−π/2,π/2] radians.

For finite values of *x* not in the range [−1,1], a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *asin*(), *asinf*(), and *asinl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is finite and is not in the range [−1,1], or is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *sin*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

asinh, asinhf, asinhl — inverse hyperbolic sine functions

SYNOPSIS

```
#include <math.h>

double asinh(double x);
float asinhf(float x);
long double asinhl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic sine of their argument.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, or ±Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *asinh*(), *asinhf*(), and *asinhl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *sinh*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

asinl — arc sine function

SYNOPSIS

```
#include <math.h>

long double asinl(long double x);
```

DESCRIPTION

Refer to *asin*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

assert — insert program diagnostics

SYNOPSIS

```
#include <assert.h>

void assert(scalar expression);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *assert*() macro shall insert diagnostics into programs; it shall expand to a **void** expression. When it is executed, if *expression* (which shall have a **scalar** type) is false (that is, compares equal to 0), *assert*() shall write information about the particular call that failed on *stderr* and shall call *abort*().

The information written about the call that failed shall include the text of the argument, the name of the source file, the source file line number, and the name of the enclosing function; the latter are, respectively, the values of the preprocessing macros __FILE__ and __LINE__ and of the identifier __func__.

Forcing a definition of the name NDEBUG, either from the compiler command line or with the preprocessor control statement **#define** NDEBUG ahead of the **#include** <*assert.h*> statement, shall stop assertions from being compiled into the program.

RETURN VALUE

The *assert*() macro shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abort( ), *stdin*

The Base Definitions volume of POSIX.1-2017, **<assert.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atan, atanf, atanl — arc tangent function

SYNOPSIS

```
#include <math.h>

double atan(double x);
float atanf(float x);
long double atanl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of *x* in the range $[-\pi/2,\pi/2]$ radians.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, $\pm\pi/2$ shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *atan*(), *atanf*(), and *atanl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atan2, atan2f, atan2l — arc tangent functions

SYNOPSIS

#include <math.h>

double atan2(double *y*, double *x*);
float atan2f(float *y*, float *x*);
long double atan2l(long double *y*, long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the principal value of the arc tangent of y/x, using the signs of both arguments to determine the quadrant of the return value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the arc tangent of y/x in the range $[-\pi,\pi]$ radians.

If y is ±0 and x is < 0, $\pm\pi$ shall be returned.

If y is ±0 and x is > 0, ±0 shall be returned.

If y is < 0 and x is ±0, $-\pi/2$ shall be returned.

If y is > 0 and x is ±0, $\pi/2$ shall be returned.

If x is 0, a pole error shall not occur.

If either x or y is NaN, a NaN shall be returned.

If the correct value would cause underflow, a range error may occur, and *atan*(), *atan2f*(), and *atan2l*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If the IEC 60559 Floating-Point option is supported, y/x should be returned.

If y is ±0 and x is −0, $\pm\pi$ shall be returned.

If y is ±0 and x is +0, ±0 shall be returned.

For finite values of $\pm y > 0$, if x is −Inf, $\pm\pi$ shall be returned.

For finite values of $\pm y > 0$, if x is +Inf, ±0 shall be returned.

For finite values of x, if y is ±Inf, $\pm\pi/2$ shall be returned.

If y is ±Inf and x is −Inf, $\pm 3\pi/4$ shall be returned.

If y is ±Inf and x is +Inf, $\pm\pi/4$ shall be returned.

If both arguments are 0, a domain error shall not occur.

ERRORS

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Converting Cartesian to Polar Coordinates System

The function below uses *atan2*() to convert a 2d vector expressed in cartesian coordinates (*x*,*y*) to the polar coordinates (*rho*,*theta*). There are other ways to compute the angle *theta*, using *asin*() *acos*(), or *atan*(). However, *atan2*() presents here two advantages:

* The angle's quadrant is automatically determined.
* The singular cases (0,*y*) are taken into account.

Finally, this example uses *hypot*() rather than *sqrt*() since it is better for special cases; see *hypot*() for more information.

```
#include <math.h>

void
cartesian_to_polar(const double x, const double y,
            double *rho, double *theta
  )
{
  *rho   = hypot (x,y); /* better than sqrt(x*x+y*y) */
  *theta = atan2 (y,x);
}
```

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acos( ), *asin*( ), *atan*( ), *feclearexcept*( ), *fetestexcept*( ), *hypot*( ), *isnan*( ), *sqrt*( ), *tan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atanf — arc tangent function

SYNOPSIS

```
#include <math.h>

float atanf(float x);
```

DESCRIPTION

Refer to *atan*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atanh, atanhf, atanhl — inverse hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double atanh(double x);
float atanhf(float x);
long double atanhl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the inverse hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the inverse hyperbolic tangent of their argument.

If *x* is ±1, a pole error shall occur, and *atanh*(), *atanhf*(), and *atanhl*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively, with the same sign as the correct value of the function.

For finite |*x*|>1, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *atanh*(), *atanhf*(), and *atanhl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Domain Error
: The *x* argument is finite and not in the range [−1,1], or is ±Inf.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error
: The *x* argument is ±1.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *tanh*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atanl — arc tangent function

SYNOPSIS

```
#include <math.h>

long double atanl(long double x);
```

DESCRIPTION

Refer to *atan*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atexit — register a function to run at process termination

SYNOPSIS

```
#include <stdlib.h>

int atexit(void (*func)(void));
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *atexit*() function shall register the function pointed to by *func*, to be called without arguments at normal program termination. At normal program termination, all functions registered by the *atexit*() function shall be called, in the reverse order of their registration, except that a function is called after any previously registered functions that had already been called at the time it was registered. Normal termination occurs either by a call to *exit*() or a return from *main*().

At least 32 functions can be registered with *atexit*().

After a successful call to any of the *exec* functions, any functions previously registered by *atexit*() shall no longer be registered.

RETURN VALUE

Upon successful completion, *atexit*() shall return 0; otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The functions registered by a call to *atexit*() must return to ensure that all registered functions are called.

The application should call *sysconf*() to obtain the value of {ATEXIT_MAX}, the number of functions that can be registered. There is no way for an application to tell how many functions have already been registered with *atexit*().

Since the behavior is undefined if the *exit*() function is called more than once, portable applications calling *atexit*() must ensure that the *exit*() function is not called at normal process termination when all functions registered by the *atexit*() function are called.

All functions registered by the *atexit*() function are called at normal process termination, which occurs by a call to the *exit*() function or a return from *main*() or on the last thread termination, when the behavior is as if the implementation called *exit*() with a zero argument at thread termination time.

If, at normal process termination, a function registered by the *atexit*() function is called and a portable application needs to stop further *exit*() processing, it must call the *_exit*() function or the *_Exit*() function or one of the functions which cause abnormal process termination.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atof — convert a string to a double-precision number

SYNOPSIS

```
#include <stdlib.h>

double atof(const char *str);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The call *atof*(*str*) shall be equivalent to:

```
strtod(str,(char **)NULL),
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

The *atof*() function shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *atof*() function is subsumed by *strtod*() but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtod*() should be used because *atof*() is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtod( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atoi — convert a string to an integer

SYNOPSIS

```
#include <stdlib.h>

int atoi(const char *str);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The call *atoi*(*str*) shall be equivalent to:

```
(int) strtol(str, (char **)NULL, 10)
```

except that the handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

The *atoi*() function shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Converting an Argument

The following example checks for proper usage of the program. If there is an argument and the decimal conversion of this argument (obtained using *atoi*()) is greater than 0, then the program has a valid number of minutes to wait for an event.

```
#include <stdlib.h>
#include <stdio.h>
...
int minutes_to_event;
...
if (argc < 2 || ((minutes_to_event = atoi (argv[1]))) <= 0) {
  fprintf(stderr, "Usage: %s minutes\n", argv[0]); exit(1);
}
...
```

APPLICATION USAGE

The *atoi*() function is subsumed by *strtol*() but is retained because it is used extensively in existing code. If the number is not known to be in range, *strtol*() should be used because *atoi*() is not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

atol, atoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long atol(const char *nptr);
long long atoll(const char *nptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

Except as noted below, the call *atol*(*nptr*) shall be equivalent to:

```
strtol(nptr, (char **)NULL, 10)
```

Except as noted below, the call to *atoll*(*nptr*) shall be equivalent to:

```
strtoll(nptr, (char **)NULL, 10)
```

The handling of errors may differ. If the value cannot be represented, the behavior is undefined.

RETURN VALUE

These functions shall return the converted value if the value can be represented.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the number is not known to be in range, *strtol*() or *strtoll*() should be used because *atol*() and *atoll*() are not required to perform any error checking.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

basename — return the last component of a pathname

SYNOPSIS

```
#include <libgen.h>

char *basename(char *path);
```

DESCRIPTION

The *basename*() function shall take the pathname pointed to by *path* and return a pointer to the final component of the pathname, deleting any trailing **'/'** characters.

If the string pointed to by *path* consists entirely of the **'/'** character, *basename*() shall return a pointer to the string **"/"**. If the string pointed to by *path* is exactly **"//"**, it is implementation-defined whether **'/'** or **"//"** is returned.

If *path* is a null pointer or points to an empty string, *basename*() shall return a pointer to the string **"."**.

The *basename*() function may modify the string pointed to by *path*, and may return a pointer to internal storage. The returned pointer might be invalidated or the storage might be overwritten by a subsequent call to *basename*(). The returned pointer might also be invalidated if the calling thread is terminated.

The *basename*() function need not be thread-safe.

RETURN VALUE

The *basename*() function shall return a pointer to the final component of *path*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Using basename( )

The following program fragment returns a pointer to the value *lib*, which is the base name of **/usr/lib**.

```
#include <libgen.h>
...
char name[] = "/usr/lib";
char *base;

base = basename(name);
...
```

Sample Input and Output Strings for the basename( ) and dirname( ) Functions and the basename and dirname Utilities

.TS center box tab(!); cB | cB | cB | cB | cB | cB cB | cB | cB | cB | cB | cB cB | cB | cB | cB | cB | cB cB | cB | cB | cB | cB | cB lf51 | lf51 | lf51 | lf51 | lf51 | lf5. basename( )!!!basename!Output!Output and dirname( )!String!String!and dirname!Written by!Written by Functions path!Returned by!Returned by!Utilities!basename!dirname Argument!basename( )!dirname( )!string Operand!Utility!Utility _ "usr"!"usr"!"."!usr!usr!. _ "usr/"!"usr"!"."!usr/!usr!. _ ""!"."!"."!""!. or empty string!. _ "/"!"/"!"/"!/!/!/ _ "//"!"/" or "//"!"/" or "//"!//!/ or //!/ or // _ "///"!"/"!"/"!///!/!/ _ "/usr/"!"usr"!"/"!/usr/!usr!/ _ "/usr/lib"!"lib"!"/usr"!/usr/lib!lib!/usr _ "//usr//lib//"!"lib"!"//usr"!//usr//lib//!lib!//usr _ "/home//dwc//!"test"!"/home//dwc"!/home//dwc//!test!/home//dwc test"!!!test .SH "APPLICATION USAGE" None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dirname( )

The Base Definitions volume of POSIX.1-2017, **<libgen.h>**

The Shell and Utilities volume of POSIX.1-2017, *basename*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

bind — bind a name to a socket

SYNOPSIS

```
#include <sys/socket.h>

int bind(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

DESCRIPTION

The *bind*() function shall assign a local socket address *address* to a socket identified by descriptor *socket* that has no local socket address assigned. Sockets created with the *socket*() function are initially unnamed; they are identified only by their address family.

The *bind*() function takes the following arguments:

socket	Specifies the file descriptor of the socket to be bound.
address	Points to a **sockaddr** structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
address_len	Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

The socket specified by *socket* may require the process to have appropriate privileges to use the *bind*() function.

If the address family of the socket is AF_UNIX and the pathname in *address* names a symbolic link, *bind*() shall fail and set *errno* to **[EADDRINUSE]**.

If the socket address cannot be assigned immediately and O_NONBLOCK is set for the file descriptor for the socket, *bind*() shall fail and set *errno* to **[EINPROGRESS]**, but the assignment request shall not be aborted, and the assignment shall be completed asynchronously. Subsequent calls to *bind*() for the same socket, before the assignment is completed, shall fail and set *errno* to **[EALREADY]**.

When the assignment has been performed asynchronously, *pselect*(), *select*(), and *poll*() shall indicate that the file descriptor for the socket is ready for reading and writing.

RETURN VALUE

Upon successful completion, *bind*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *bind*() function shall fail if:

EADDRINUSE
The specified address is already in use.

EADDRNOTAVAIL
The specified address is not available from the local machine.

EAFNOSUPPORT
The specified address is not a valid address for the address family of the specified socket.

EALREADY
An assignment request is already in progress for the specified socket.

EBADF
The *socket* argument is not a valid file descriptor.

EINPROGRESS
: O_NONBLOCK is set for the file descriptor for the socket and the assignment cannot be immediately performed; the assignment shall be performed asynchronously.

EINVAL
: The socket is already bound to an address, and the protocol does not support binding to a new address; or the socket has been shut down.

ENOBUFS
: Insufficient resources were available to complete the call.

ENOTSOCK
: The *socket* argument does not refer to a socket.

EOPNOTSUPP
: The socket type of the specified socket does not support binding to an address.

If the address family of the socket is AF_UNIX, then *bind*() shall fail if:

EACCES
: A component of the path prefix denies search permission, or the requested name requires writing in a directory with a mode that denies write permission.

EDESTADDRREQ or **EISDIR**
: The *address* argument is a null pointer.

EIO An I/O error occurred.

ELOOP
: A loop exists in symbolic links encountered during resolution of the pathname in *address*.

ENAMETOOLONG
: The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
: A component of the path prefix of the pathname in *address* does not name an existing file or the pathname is an empty string.

ENOENT or **ENOTDIR**
: The pathname in *address* contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If the pathname without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOTDIR
: A component of the path prefix of the pathname in *address* names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in *address* contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
: The name would reside on a read-only file system.

The *bind*() function may fail if:

EACCES
: The specified address is protected and the current user does not have permission to bind to it.

EINVAL
: The *address_len* argument is not a valid length for the address family.

EISCONN
: The socket is already connected.

ELOOP
: More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in *address*.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

The following code segment shows how to create a socket and bind it to a name in the AF_UNIX domain.

```
#define MY_SOCK_PATH "/somepath"

int sfd;
struct sockaddr_un my_addr;

sfd = socket(AF_UNIX, SOCK_STREAM, 0);
if (sfd == -1)
    /* Handle error */;

memset(&my_addr, '\0', sizeof(struct sockaddr_un));
                /* Clear structure */
my_addr.sun_family = AF_UNIX;
strncpy(my_addr.sun_path, MY_SOCK_PATH, sizeof(my_addr.sun_path) -1);

if (bind(sfd, (struct sockaddr *) &my_addr,
    sizeof(struct sockaddr_un)) == -1)
    /* Handle error */;
```

APPLICATION USAGE

An application program can retrieve the assigned socket name with the *getsockname*() function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *getsockname*( ), *listen*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

bsearch — binary search a sorted table

SYNOPSIS

```
#include <stdlib.h>

void *bsearch(const void *key, const void *base, size_t nel,
   size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *bsearch*() function shall search an array of *nel* objects, the initial element of which is pointed to by *base*, for an element that matches the object pointed to by *key*. The size of each element in the array is specified by *width*. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no match shall be found.

The comparison function pointed to by *compar* shall be called with two arguments that point to the *key* object and to an array element, in that order.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

The implementation shall ensure that the first argument is always a pointer to the key.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, the same object shall always compare the same way with the key.

The application shall ensure that the function returns an integer less than, equal to, or greater than 0 if the *key* object is considered, respectively, to be less than, to match, or to be greater than the array element. The application shall ensure that the array consists of all the elements that compare less than, all the elements that compare equal to, and all the elements that compare greater than the *key* object, in that order.

RETURN VALUE

The *bsearch*() function shall return a pointer to a matching member of the array, or a null pointer if no match is found. If two or more members compare equal, which member is returned is unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The example below searches a table containing pointers to nodes consisting of a string and its length. The table is ordered alphabetically on the string in the node pointed to by each entry.

The code fragment below reads in strings and either finds the corresponding node and prints out the string and its length, or prints an error message.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define TABSIZE    1000
```

```
struct node {              /* These are stored in the table. */
    char *string;
    int length;
};
struct node table[TABSIZE];    /* Table to be searched. */
    .
    .
    .
{
    struct node *node_ptr, node;
    /* Routine to compare 2 nodes. */
    int node_compare(const void *, const void *);
    .
    .
    .
    while (scanf("%ms", &node.string) != EOF) {
        node_ptr = (struct node *)bsearch((void *)(&node),
               (void *)table, TABSIZE,
               sizeof(struct node), node_compare);
        if (node_ptr != NULL) {
            (void)printf("string = %20s, length = %d\n",
                node_ptr->string, node_ptr->length);
        } else {
            (void)printf("not found: %s\n", node.string);
        }
        free(node.string);
    }
}
/*
    This routine compares two nodes based on an
    alphabetical ordering of the string field.
*/
int
node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
```

APPLICATION USAGE

The pointers to the key and the element at the base of the table should be of type pointer-to-element.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

In practice, the array is usually sorted according to the comparison function.

RATIONALE

The requirement that the second argument (hereafter referred to as *p*) to the comparison function is a pointer to an element of the array implies that for every call all of the following expressions are non-zero:

```
( (char *)p - (char *)base ) % width == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *lsearch*( ), *qsort*( ), *tdelete*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

btowc — single byte to wide character conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t btowc(int c);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *btowc*() function shall determine whether *c* constitutes a valid (one-byte) character in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *btowc*() function shall return WEOF if *c* has the value EOF or if **(unsigned char)** *c* does not constitute a valid (one-byte) character in the initial shift state. Otherwise, it shall return the wide-character representation of that character.

In the POSIX locale, *btowc*() shall not return WEOF if *c* has a value in the range 0 to 255 inclusive.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wctob( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cabs, cabsf, cabsl — return a complex absolute value

SYNOPSIS

```
#include <complex.h>

double cabs(double complex z);
float cabsf(float complex z);
long double cabsl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex absolute value (also called norm, modulus, or magnitude) of *z*.

RETURN VALUE

These functions shall return the complex absolute value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cacos, cacosf, cacosl — complex arc cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacos(double complex z);
float complex cacosf(float complex z);
long double complex cacosl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc cosine of *z*, with branch cuts outside the interval [−1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc cosine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [0, *π*] along the real axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccos( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cacosh, cacoshf, cacoshl — complex arc hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex cacosh(double complex z);
float complex cacoshf(float complex z);
long double complex cacoshl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic cosine of *z*, with a branch cut at values less than 1 along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic cosine value, in the range of a half-strip of non-negative values along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ccosh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cacosl — complex arc cosine functions

SYNOPSIS

#include <complex.h>

long double complex cacosl(long double complex *z*);

DESCRIPTION

Refer to *cacos*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

calloc — a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *calloc(size_t nelem, size_t elsize);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *calloc*() function shall allocate unused space for an array of *nelem* elements each of whose size in bytes is *elsize*. The space shall be initialized to all bits 0.

The order and contiguity of storage allocated by successive calls to *calloc*() is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object or an array of such objects in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the returned pointer is used to access an object.

RETURN VALUE

Upon successful completion with both *nelem* and *elsize* non-zero, *calloc*() shall return a pointer to the allocated space. If either *nelem* or *elsize* is 0, then either:

* A null pointer shall be returned and *errno* may be set to an implementation-defined value, or
* A pointer to the allocated space shall be returned. The application shall ensure that the pointer is not used to access an object.

Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *calloc*() function shall fail if:

ENOMEM
Insufficient memory is available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of *<malloc.h>*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *malloc*( ), *realloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

carg, cargf, cargl — complex argument functions

SYNOPSIS

#include <complex.h>

double carg(double complex *z*);
float cargf(float complex *z*);
long double cargl(long double complex *z*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the argument (also called phase angle) of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the value of the argument in the interval $[-\pi, +\pi]$.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cimag( ), *conj*( ), *cproj*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

casin, casinf, casinl — complex arc sine functions

SYNOPSIS

```
#include <complex.h>

double complex casin(double complex z);
float complex casinf(float complex z);
long double complex casinl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc sine of *z*, with branch cuts outside the interval [−1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc sine value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [−*π*/2, +*π*/2] along the real axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

csin( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

casinh, casinhf, casinhl — complex arc hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex casinh(double complex z);
float complex casinhf(float complex z);
long double complex casinhl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic sine of *z*, with branch cuts outside the interval $[-i, +i]$ along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic sine value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

csinh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

casinl — complex arc sine functions

SYNOPSIS

#include <complex.h>

long double complex casinl(long double complex *z*);

DESCRIPTION

Refer to *casin*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catan, catanf, catanl — complex arc tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catan(double complex z);
float complex catanf(float complex z);
long double complex catanl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc tangent of *z*, with branch cuts outside the interval [−*i*, +*i*] along the imaginary axis.

RETURN VALUE

These functions shall return the complex arc tangent value, in the range of a strip mathematically unbounded along the imaginary axis and in the interval [$-\pi/2$, $+\pi/2$] along the real axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctan( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catanh, catanhf, catanhl — complex arc hyperbolic tangent functions

SYNOPSIS

```
#include <complex.h>

double complex catanh(double complex z);
float complex catanhf(float complex z);
long double complex catanhl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex arc hyperbolic tangent of *z*, with branch cuts outside the interval [−1, +1] along the real axis.

RETURN VALUE

These functions shall return the complex arc hyperbolic tangent value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi/2, +i\pi/2]$ along the imaginary axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctanh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catanl — complex arc tangent functions

SYNOPSIS

```
#include <complex.h>

long double complex catanl(long double complex z);
```

DESCRIPTION

Refer to *catan*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catclose — close a message catalog descriptor

SYNOPSIS

```
#include <nl_types.h>

int catclose(nl_catd catd);
```

DESCRIPTION

The *catclose*() function shall close the message catalog identified by *catd*. If a file descriptor is used to implement the type **nl_catd**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, *catclose*() shall return 0; otherwise, −1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *catclose*() function may fail if:

EBADF
The catalog descriptor is not valid.

EINTR
The *catclose*() function was interrupted by a signal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catgets( ), *catopen*( )

The Base Definitions volume of POSIX.1-2017, **<nl_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catgets — read a program message

SYNOPSIS

#include <nl_types.h>

char *catgets(nl_catd *catd*, int *set_id*, int *msg_id*, const char **s*);

DESCRIPTION

The *catgets*() function shall attempt to read message *msg_id*, in set *set_id*, from the message catalog identified by *catd*. The *catd* argument is a message catalog descriptor returned from an earlier call to *catopen*(). The results are undefined if *catd* is not a value returned by *catopen*() for a message catalog still open in the process. The *s* argument points to a default message string which shall be returned by *catgets*() if it cannot retrieve the identified message.

The *catgets*() function need not be thread-safe.

RETURN VALUE

If the identified message is retrieved successfully, *catgets*() shall return a pointer to an internal buffer area containing the null-terminated message string. If the call is unsuccessful for any reason, *s* shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *catgets*() function shall fail if:

EINTR
The read operation was terminated due to the receipt of a signal, and no data was transferred.

ENOMSG
The message identified by *set_id* and *msg_id* is not in the message catalog.

The *catgets*() function may fail if:

EBADF
The *catd* argument is not a valid message catalog descriptor open for reading.

EBADMSG
The message identified by *set_id* and *msg_id* in the specified message catalog did not satisfy implementation-defined security criteria.

EINVAL
The message catalog identified by *catd* is corrupted.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catopen*( )

The Base Definitions volume of POSIX.1-2017, **<nl_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

catopen — open a message catalog

SYNOPSIS

```
#include <nl_types.h>

nl_catd catopen(const char *name, int oflag);
```

DESCRIPTION

The *catopen*() function shall open a message catalog and return a message catalog descriptor. The *name* argument specifies the name of the message catalog to be opened. If *name* contains a **'/'**, then *name* specifies a pathname for the message catalog. Otherwise, the environment variable *NLSPATH* is used with *name* substituted for the **%N** conversion specification (see the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*); if *NLSPATH* exists in the environment when the process starts, then if the process has appropriate privileges, the behavior of *catopen*() is undefined. If *NLSPATH* does not exist in the environment, or if a message catalog cannot be found in any of the components specified by *NLSPATH*, then an implementation-defined default path shall be used. This default may be affected by the setting of *LC_MESSAGES* if the value of *oflag* is NL_CAT_LOCALE, or the *LANG* environment variable if *oflag* is 0.

A message catalog descriptor shall remain valid in a process until that process closes it, or a successful call to one of the *exec* functions. A change in the setting of the *LC_MESSAGES* category may invalidate existing open catalogs.

If a file descriptor is used to implement message catalog descriptors, the FD_CLOEXEC flag shall be set; see *<fcntl.h>*.

If the value of the *oflag* argument is 0, the *LANG* environment variable is used to locate the catalog without regard to the *LC_MESSAGES* category. If the *oflag* argument is NL_CAT_LOCALE, the *LC_MESSAGES* category is used to locate the message catalog (see the Base Definitions volume of POSIX.1-2017, *Section 8.2*, *Internationalization Variables*).

RETURN VALUE

Upon successful completion, *catopen*() shall return a message catalog descriptor for use on subsequent calls to *catgets*() and *catclose*(). Otherwise, *catopen*() shall return (**nl_catd**) −1 and set *errno* to indicate the error.

ERRORS

The *catopen*() function may fail if:

EACCES
Search permission is denied for the component of the path prefix of the message catalog or read permission is denied for the message catalog.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE
Too many files are currently open in the system.

ENOENT
The message catalog does not exist or the *name* argument points to an empty string.

ENOMEM
Insufficient storage space is available.

ENOTDIR
A component of the path prefix of the message catalog names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname of the message catalog contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Some implementations of *catopen*() use *malloc*() to allocate space for internal buffer areas. The *catopen*() function may fail if there is insufficient storage space available to accommodate these buffers.

Conforming applications must assume that message catalog descriptors are not valid after a call to one of the *exec* functions.

Application developers should be aware that guidelines for the location of message catalogs have not yet been developed. Therefore they should take care to avoid conflicting with catalogs used by other applications and the standard utilities.

To be sure that messages produced by an application running with appropriate privileges cannot be used by an attacker setting an unexpected value for *NLSPATH* in the environment to confuse a system administrator, such applications should use pathnames containing a **'/'** to get defined behavior when using *catopen*() to open a message catalog.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catclose( ), *catgets*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*, **<fcntl.h>**, **<nl_types.h>**,

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cbrt, cbrtf, cbrtl — cube root functions

SYNOPSIS

```
#include <math.h>

double cbrt(double x);
float cbrtf(float x);
long double cbrtl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the real cube root of their argument *x*.

RETURN VALUE

Upon successful completion, these functions shall return the cube root of *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

For some applications, a true cube root function, which returns negative results for negative arguments, is more appropriate than *pow*(*x*, 1.0/3.0), which returns a NaN for *x* less than 0.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ccos, ccosf, ccosl — complex cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccos(double complex z);
float complex ccosf(float complex z);
long double complex ccosl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex cosine of *z*.

RETURN VALUE

These functions shall return the complex cosine value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cacos( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ccosh, ccoshf, ccoshl — complex hyperbolic cosine functions

SYNOPSIS

```
#include <complex.h>

double complex ccosh(double complex z);
float complex ccoshf(float complex z);
long double complex ccoshl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex hyperbolic cosine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic cosine value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cacosh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ccosl — complex cosine functions

SYNOPSIS

```
#include <complex.h>

long double complex ccosl(long double complex z);
```

DESCRIPTION

Refer to *ccos*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ceil, ceilf, ceill — ceiling value function

SYNOPSIS

```
#include <math.h>

double ceil(double x);
float ceilf(float x);
long double ceill(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the smallest integral value not less than *x*.

RETURN VALUE

The result shall have the same sign as *x*.

Upon successful completion, *ceil*(), *ceilf*(), and *ceill*() shall return the smallest integral value not less than *x*, expressed as a type **double**, **float**, or **long double**, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

These functions may raise the inexact floating-point exception if the result differs in value from the argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *floor*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cexp, cexpf, cexpl — complex exponential functions

SYNOPSIS

```
#include <complex.h>

double complex cexp(double complex z);
float complex cexpf(float complex z);
long double complex cexpl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex exponent of *z*, defined as e^z.

RETURN VALUE

These functions shall return the complex exponential value of *z*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clog( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cfgetispeed — get input baud rate

SYNOPSIS

#include <termios.h>

speed_t cfgetispeed(const struct termios **termios_p*);

DESCRIPTION

The *cfgetispeed*() function shall extract the input baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetispeed*() shall return a value of type **speed_t** representing the input baud rate.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term "baud" is used historically here, but is not technically correct. This is properly "bits per second", which may not be the same as baud. However, the term is used because of the historical usage and understanding.

The *cfgetospeed*(), *cfgetispeed*(), *cfsetospeed*(), and *cfsetispeed*() functions do not take arguments as numbers, but rather as symbolic names. There are two reasons for this:

1. Historically, numbers were not used because of the way the rate was stored in the data structure. This is retained even though a function is now used.
2. More importantly, only a limited set of possible rates is at all portable, and this constrains the application to that set.

There is nothing to prevent an implementation accepting as an extension a number (such as 126), and since the encoding of the Bxxx symbols is not specified, this can be done to avoid introducing ambiguity.

Setting the input baud rate to zero was a mechanism to allow for split baud rates. Clarifications in this volume of POSIX.1-2017 have made it possible to determine whether split rates are supported and to support them without having to treat zero as a special case. Since this functionality is also confusing, it has been declared obsolescent. The 0 argument referred to is the literal constant 0, not the symbolic constant B0. This volume of POSIX.1-2017 does not preclude B0 from being defined as the value 0; in fact, implementations would likely benefit from the two being equivalent. This volume of POSIX.1-2017 does not fully specify whether the previous *cfsetispeed*() value is retained after a *tcgetattr*() as the actual value or as zero. Therefore, conforming applications should always set both the input speed and output speed when setting either.

In historical implementations, the baud rate information is traditionally kept in **c_cflag**. Applications should be written to presume that this might be the case (and thus not blindly copy **c_cflag**), but not to rely on it in case it is in some other field of the structure. Setting the **c_cflag** field absolutely after setting a baud rate is a non-portable action because of this. In general, the unused parts of the flag fields might be used by the implementation and should not be blindly copied from the descriptions of one terminal device to another.

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetospeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cfgetospeed — get output baud rate

SYNOPSIS

```
#include <termios.h>

speed_t cfgetospeed(const struct termios *termios_p);
```

DESCRIPTION

The *cfgetospeed*() function shall extract the output baud rate from the **termios** structure to which the *termios_p* argument points.

This function shall return exactly the value in the **termios** data structure, without interpretation.

RETURN VALUE

Upon successful completion, *cfgetospeed*() shall return a value of type **speed_t** representing the output baud rate.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfsetispeed*( ), *cfsetospeed*( ), *tcgetattr*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cfsetispeed — set input baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetispeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetispeed*() function shall set the input baud rate stored in the structure pointed to by *termios_p* to *speed.*

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*() with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*() function is called.

RETURN VALUE

Upon successful completion, *cfsetispeed*() shall return 0; otherwise, −1 shall be returned, and *errno* may be set to indicate the error.

ERRORS

The *cfsetispeed*() function may fail if:

EINVAL
The *speed* value is not a valid baud rate.

EINVAL
The value of *speed* is outside the range of possible speed values as specified in *<termios.h>*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetospeed*( ), *tcsetattr*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cfsetospeed — set output baud rate

SYNOPSIS

```
#include <termios.h>

int cfsetospeed(struct termios *termios_p, speed_t speed);
```

DESCRIPTION

The *cfsetospeed*() function shall set the output baud rate stored in the structure pointed to by *termios_p* to *speed*.

There shall be no effect on the baud rates set in the hardware until a subsequent successful call to *tcsetattr*() with the same **termios** structure. Similarly, errors resulting from attempts to set baud rates not supported by the terminal device need not be detected until the *tcsetattr*() function is called.

RETURN VALUE

Upon successful completion, *cfsetospeed*() shall return 0; otherwise, it shall return −1 and *errno* may be set to indicate the error.

ERRORS

The *cfsetospeed*() function may fail if:

EINVAL
The *speed* value is not a valid baud rate.

EINVAL
The value of *speed* is outside the range of possible speed values as specified in *<termios.h>*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *cfgetispeed*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

cfgetispeed( ), *cfgetospeed*( ), *cfsetispeed*( ), *tcsetattr*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

chdir — change working directory

SYNOPSIS

```
#include <unistd.h>

int chdir(const char *path);
```

DESCRIPTION

The *chdir*() function shall cause the directory named by the pathname pointed to by the *path* argument to become the current working directory; that is, the starting point for path searches for pathnames not beginning with **'/'**.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned, the current working directory shall remain unchanged, and *errno* shall be set to indicate the error.

ERRORS

The *chdir*() function shall fail if:

EACCES
Search permission is denied for any component of the pathname.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing directory or *path* is an empty string.

ENOTDIR
A component of the pathname names an existing file that is neither a directory nor a symbolic link to a directory.

The *chdir*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Changing the Current Working Directory

The following example makes the value pointed to by **directory**, **/tmp**, the current working directory.

```
#include <unistd.h>
...
char *directory = "/tmp";
int ret;

ret = chdir (directory);
```

APPLICATION USAGE

None.

RATIONALE

The *chdir*() function only affects the working directory of the current process.

FUTURE DIRECTIONS

None.

SEE ALSO

getcwd( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

chmod, fchmodat — change mode of a file

SYNOPSIS

#include <sys/stat.h>

int chmod(const char **path*, mode_t *mode*);

#include <fcntl.h>

int fchmodat(int *fd*, const char **path*, mode_t *mode*, int *flag*);

DESCRIPTION

The *chmod*() function shall change S_ISUID, S_ISGID, S_ISVTX, and the file permission bits of the file named by the pathname pointed to by the *path* argument to the corresponding bits in the *mode* argument. The application shall ensure that the effective user ID of the process matches the owner of the file or the process has appropriate privileges in order to do this.

S_ISUID, S_ISGID, S_ISVTX, and the file permission bits are described in *<sys/stat.h>*.

If the calling process does not have appropriate privileges, and if the group ID of the file does not match the effective group ID or one of the supplementary group IDs and if the file is a regular file, bit S_ISGID (set-group-ID on execution) in the file's mode shall be cleared upon successful return from *chmod*().

Additional implementation-defined restrictions may cause the S_ISUID and S_ISGID bits in *mode* to be ignored.

Upon successful completion, *chmod*() shall mark for update the last file status change timestamp of the file.

The *fchmodat*() function shall be equivalent to the *chmod*() function except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_SYMLINK_NOFOLLOW
    If *path* names a symbolic link, then the mode of the symbolic link is changed.

If *fchmodat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used. If also *flag* is zero, the behavior shall be identical to a call to *chmod*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, no change to the file mode occurs.

ERRORS

These functions shall fail if:

EACCES
    Search permission is denied on a component of the path prefix.

ELOOP
    A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
    The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

EROFS
The named file resides on a read-only file system.

The *fchmodat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

EINTR
A signal was caught during execution of the function.

EINVAL
The value of the *mode* argument is invalid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *fchmodat*() function may fail if:

EINVAL
The value of the *flag* argument is invalid.

EOPNOTSUPP
The AT_SYMLINK_NOFOLLOW bit is set in the *flag* argument, *path* names a symbolic link, and the system does not support changing the mode of a symbolic link.

The following sections are informative.

EXAMPLES

Setting Read Permissions for User, Group, and Others

The following example sets read permissions for the owner, group, and others.

```
#include <sys/stat.h>
const char *path;
```

```
...
chmod(path, S_IRUSR|S_IRGRP|S_IROTH);
```

Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

```
#include <sys/stat.h>

const char *path;
...
chmod(path, S_IRWXU);
```

Setting Different Permissions for Owner, Group, and Other

The following example sets owner permissions for CHANGEFILE to read, write, and execute, group permissions to read and execute, and other permissions to read.

```
#include <sys/stat.h>

#define CHANGEFILE "/etc/myfile"
...
chmod(CHANGEFILE, S_IRWXU|S_IRGRP|S_IXGRP|S_IROTH);
```

Setting and Checking File Permissions

The following example sets the file permission bits for a file named **/home/cnd/mod1**, then calls the *stat*() function to verify the permissions.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
struct stat buffer
...
chmod("/home/cnd/mod1", S_IRWXU|S_IRWXG|S_IROTH|S_IWOTH);
status = stat("/home/cnd/mod1", &buffer);
```

APPLICATION USAGE

In order to ensure that the S_ISUID and S_ISGID bits are set, an application requiring this should use *stat*() after a successful *chmod*() to verify this.

Any file descriptors currently open by any process on the file could possibly become invalid if the mode of the file is changed to a value which would deny access to that process. One situation where this could occur is on a stateless file system. This behavior will not occur in a conforming environment.

RATIONALE

This volume of POSIX.1-2017 specifies that the S_ISGID bit is cleared by *chmod*() on a regular file under certain conditions. This is specified on the assumption that regular files may be executed, and the system should prevent users from making executable *setgid*() files perform with privileges that the caller does not have. On implementations that support execution of other file types, the S_ISGID bit should be cleared for those file types under the same circumstances.

Implementations that use the S_ISUID bit to indicate some other function (for example, mandatory record locking) on non-executable files need not clear this bit on writing. They should clear the bit for executable files and any other cases where the bit grants special powers to processes that change the file contents. Similar comments apply to the S_ISGID bit.

The purpose of the *fchmodat*() function is to enable changing the mode of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chmod*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchmodat*() function it can be guaranteed that the changed file is located relative to the desired directory. Some implementations might allow changing the mode of symbolic links. This is not supported by the interfaces in the POSIX specification. Systems with such support provide an interface named *lchmod*( ). To support such implementations *fchmodat*() has a *flag* parameter.

FUTURE DIRECTIONS

None.

SEE ALSO

access( ), *chown*( ), *exec*, *fstatat*( ), *fstatvfs*( ), *mkdir*( ), *mkfifo*( ), *mknod*( ), *open*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

chown, fchownat — change owner and group of a file

SYNOPSIS

```
#include <unistd.h>

int chown(const char *path, uid_t owner, gid_t group);

#include <fcntl.h>

int fchownat(int fd, const char *path, uid_t owner, gid_t group,
    int flag);
```

DESCRIPTION

The *chown*() function shall change the user and group ownership of a file.

The *path* argument points to a pathname naming a file. The user ID and group ID of the named file shall be set to the numeric values contained in *owner* and *group*, respectively.

Only processes with an effective user ID equal to the user ID of the file or with appropriate privileges may change the ownership of a file. If _POSIX_CHOWN_RESTRICTED is in effect for *path*:

* Changing the user ID is restricted to processes with appropriate privileges.

* Changing the group ID is permitted to a process with an effective user ID equal to the user ID of the file, but without appropriate privileges, if and only if *owner* is equal to the file's user ID or (**uid_t**)−1 and *group* is equal either to the calling process' effective group ID or to one of its supplementary group IDs.

If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set, and the process does not have appropriate privileges, the set-user-ID (S_ISUID) and set-group-ID (S_ISGID) bits of the file mode shall be cleared upon successful return from *chown*(). If the specified file is a regular file, one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set, and the process has appropriate privileges, it is implementation-defined whether the set-user-ID and set-group-ID bits are altered. If the *chown*() function is successfully invoked on a file that is not a regular file and one or more of the S_IXUSR, S_IXGRP, or S_IXOTH bits of the file mode are set, the set-user-ID and set-group-ID bits may be cleared.

If *owner* or *group* is specified as (**uid_t**)−1 or (**gid_t**)−1, respectively, the corresponding ID of the file shall not be changed.

Upon successful completion, *chown*() shall mark for update the last file status change timestamp of the file, except that if *owner* is (**uid_t**)−1 and *group* is (**gid_t**)−1, the file status change timestamp need not be marked for update.

The *fchownat*() function shall be equivalent to the *chown*() and *lchown*() functions except in the case where *path* specifies a relative path. In this case the file to be changed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, ownership of the symbolic link is changed.

If *fchownat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *chown*() or *lchown*() respectively, depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in the *flag* argument.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, no changes are made in the user ID and group ID of the file.

ERRORS

These functions shall fail if:

EACCES
Search permission is denied on a component of the path prefix.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The effective user ID does not match the owner of the file, or the calling process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

EROFS
The named file resides on a read-only file system.

The *fchownat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

EIO An I/O error occurred while reading or writing to the file system.

EINTR
The *chown*() function was interrupted by a signal which was caught.

EINVAL
The owner or group ID supplied is not a value supported by the implementation.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *fchownat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Although *chown*() can be used on some implementations by the file owner to change the owner and group to any desired values, the only portable use of this function is to change the group of a file to the effective GID of the calling process or to a member of its group set.

RATIONALE

System III and System V allow a user to give away files; that is, the owner of a file may change its user ID to anything. This is a serious problem for implementations that are intended to meet government security regulations. Version 7 and 4.3 BSD permit only the superuser to change the user ID of a file. Some government agencies (usually not ones concerned directly with security) find this limitation too confining. This volume of POSIX.1-2017 uses *may* to permit secure implementations while not disallowing System V.

System III and System V allow the owner of a file to change the group ID to anything. Version 7 permits only the superuser to change the group ID of a file. 4.3 BSD permits the owner to change the group ID of a file to its effective group ID or to any of the groups in the list of supplementary group IDs, but to no others.

The POSIX.1-1990 standard requires that the *chown*() function invoked by a non-appropriate privileged process clear the S_ISGID and the S_ISUID bits for regular files, and permits them to be cleared for other types of files. This is so that changes in accessibility do not accidentally cause files to become security holes. Unfortunately, requiring these bits to be cleared on non-executable data files also clears the mandatory file locking bit (shared with S_ISGID), which is an extension on many implementations (it first appeared in System V). These bits should only be required to be cleared on regular files that have one or more of their execute bits set.

The purpose of the *fchownat*() function is to enable changing ownership of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *chown*() or *lchown*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fchownat*() function it can be guaranteed that the changed file is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *fpathconf*( ), *lchown*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cimag, cimagf, cimagl — complex imaginary functions

SYNOPSIS

```
#include <complex.h>

double cimag(double complex z);
float cimagf(float complex z);
long double cimagl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the imaginary part of *z*.

RETURN VALUE

These functions shall return the imaginary part value (as a real).

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of complex type:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *conj*( ), *cproj*( ), *creal*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clearerr — clear indicators on a stream

SYNOPSIS

#include <stdio.h>

void clearerr(FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *clearerr*() function shall clear the end-of-file and error indicators for the stream to which *stream* points.

The *clearerr*() function shall not change the setting of *errno* if *stream* is valid.

RETURN VALUE

The *clearerr*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clock — report CPU time used

SYNOPSIS

```
#include <time.h>

clock_t clock(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *clock*() function shall return the implementation's best approximation to the processor time used by the process since the beginning of an implementation-defined era related only to the process invocation.

RETURN VALUE

To determine the time in seconds, the value returned by *clock*() should be divided by the value of the macro CLOCKS_PER_SEC. CLOCKS_PER_SEC is defined to be one million in <*time.h*>. If the processor time used is not available or its value cannot be represented, the function shall return the value (**clock_t**)−1.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

In programming environments where **clock_t** is a 32-bit integer type and CLOCKS_PER_SEC is one million, *clock*() will start failing in less than 36 minutes of processor time for signed **clock_t**, or 72 minutes for unsigned **clock_t**. Applications intended to be portable to such environments should use *times*() instead (or *clock_gettime*() with CLOCK_PROCESS_CPUTIME_ID, if supported).

In order to measure the time spent in a program, *clock*() should be called at the start of the program and its return value subtracted from the value returned by subsequent calls. The value returned by *clock*() is defined for compatibility across systems that have clocks with different resolutions. The resolution on any particular system need not be to microsecond accuracy.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock_getres*( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *times*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clock_getcpuclockid — access a process CPU-time clock (**ADVANCED REALTIME**)

SYNOPSIS

#include <time.h>

int clock_getcpuclockid(pid_t *pid*, clockid_t **clock_id*);

DESCRIPTION

The *clock_getcpuclockid*() function shall return the clock ID of the CPU-time clock of the process specified by *pid*. If the process described by *pid* exists and the calling process has permission, the clock ID of this clock shall be returned in *clock_id*.

If *pid* is zero, the *clock_getcpuclockid*() function shall return the clock ID of the CPU-time clock of the process making the call, in *clock_id*.

The conditions under which one process has permission to obtain the CPU-time clock ID of other processes are implementation-defined.

RETURN VALUE

Upon successful completion, *clock_getcpuclockid*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *clock_getcpuclockid*() function shall fail if:

EPERM
The requesting process does not have permission to access the CPU-time clock for the process.

The *clock_getcpuclockid*() function may fail if:

ESRCH
No process can be found corresponding to the process specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *clock_getcpuclockid*() function is part of the Process CPU-Time Clocks option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *timer_create*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clock_getres, clock_gettime, clock_settime — clock and timer functions

SYNOPSIS

```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
int clock_gettime(clockid_t clock_id, struct timespec *tp);
int clock_settime(clockid_t clock_id, const struct timespec *tp);
```

DESCRIPTION

The *clock_getres*() function shall return the resolution of any clock. Clock resolutions are implementation-defined and cannot be set by a process. If the argument *res* is not NULL, the resolution of the specified clock shall be stored in the location pointed to by *res*. If *res* is NULL, the clock resolution is not returned. If the *time* argument of *clock_settime*() is not a multiple of *res*, then the value is truncated to a multiple of *res*.

The *clock_gettime*() function shall return the current value *tp* for the specified clock, *clock_id*.

The *clock_settime*() function shall set the specified clock, *clock_id*, to the value specified by *tp*. Time values that are between two consecutive non-negative integer multiples of the resolution of the specified clock shall be truncated down to the smaller multiple of the resolution.

A clock may be system-wide (that is, visible to all processes) or per-process (measuring time that is meaningful only within a process). All implementations shall support a *clock_id* of CLOCK_REALTIME as defined in <*time.h*>. This clock represents the clock measuring real time for the system. For this clock, the values returned by *clock_gettime*() and specified by *clock_settime*() represent the amount of time (in seconds and nanoseconds) since the Epoch. An implementation may also support additional clocks. The interpretation of time values for these clocks is unspecified.

If the value of the CLOCK_REALTIME clock is set via *clock_settime*(), the new value of the clock shall be used to determine the time of expiration for absolute time services based upon the CLOCK_REALTIME clock. This applies to the time at which armed absolute timers expire. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the time service shall expire immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*() shall have no effect on threads that are blocked waiting for a relative time service based upon this clock, including the *nanosleep*() function; nor on the expiration of relative timers based upon this clock. Consequently, these time services shall expire when the requested relative interval elapses, independently of the new or old value of the clock.

If the Monotonic Clock option is supported, all implementations shall support a *clock_id* of CLOCK_MONOTONIC defined in <*time.h*>. This clock represents the monotonic clock for the system. For this clock, the value returned by *clock_gettime*() represents the amount of time (in seconds and nanoseconds) since an unspecified point in the past (for example, system start-up time, or the Epoch). This point does not change after system start-up time. The value of the CLOCK_MONOTONIC clock cannot be set via *clock_settime*(). This function shall fail if it is invoked with a *clock_id* argument of CLOCK_MONOTONIC.

The effect of setting a clock via *clock_settime*() on armed per-process timers associated with a clock other than CLOCK_REALTIME is implementation-defined.

If the value of the CLOCK_REALTIME clock is set via *clock_settime*(), the new value of the clock shall be used to determine the time at which the system shall awaken a thread blocked on an absolute *clock_nanosleep*() call based upon the CLOCK_REALTIME clock. If the absolute time requested at the invocation of such a time service is before the new value of the clock, the call shall return immediately as if the clock had reached the requested time normally.

Setting the value of the CLOCK_REALTIME clock via *clock_settime*() shall have no effect on any thread that is blocked on a relative *clock_nanosleep*() call. Consequently, the call shall return when the requested relative interval elapses, independently of the new or old value of the clock.

Appropriate privileges to set a particular clock are implementation-defined.

If _POSIX_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *clock_getcpuclockid*(), which represent the CPU-time clock of a given process. Implementations shall also support the special **clockid_t** value CLOCK_PROCESS_CPUTIME_ID, which represents the CPU-time clock of the calling process when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*() and specified by *clock_settime*() represent the amount of execution time of the process associated with the clock. Changing the value of a CPU-time clock via *clock_settime*() shall have no effect on the behavior of the sporadic server scheduling policy (see *Scheduling Policies*).

If _POSIX_THREAD_CPUTIME is defined, implementations shall support clock ID values obtained by invoking *pthread_getcpuclockid*(), which represent the CPU-time clock of a given thread. Implementations shall also support the special **clockid_t** value CLOCK_THREAD_CPUTIME_ID, which represents the CPU-time clock of the calling thread when invoking one of the *clock_**( ) or *timer_**( ) functions. For these clock IDs, the values returned by *clock_gettime*() and specified by *clock_settime*() shall represent the amount of execution time of the thread associated with the clock. Changing the value of a CPU-time clock via *clock_settime*() shall have no effect on the behavior of the sporadic server scheduling policy (see *Scheduling Policies*).

RETURN VALUE

A return value of 0 shall indicate that the call succeeded. A return value of −1 shall indicate that an error occurred, and *errno* shall be set to indicate the error.

ERRORS

The *clock_getres*(), *clock_gettime*(), and *clock_settime*() functions shall fail if:

EINVAL
The *clock_id* argument does not specify a known clock.

The *clock_gettime*() function shall fail if:

EOVERFLOW
The number of seconds will not fit in an object of type **time_t**.

The *clock_settime*() function shall fail if:

EINVAL
The *tp* argument to *clock_settime*() is outside the range for the given clock ID.

EINVAL
The *tp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

EINVAL
The value of the *clock_id* argument is CLOCK_MONOTONIC.

The *clock_settime*() function may fail if:

EPERM
The requesting process does not have appropriate privileges to set the specified clock.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Note that the absolute value of the monotonic clock is meaningless (because its origin is arbitrary), and thus there is no need to set it. Furthermore, realtime applications can rely on the fact that the value of this clock is never set and, therefore, that time intervals measured with this clock will not be affected by calls to *clock_settime*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Scheduling Policies, *clock_getcpuclockid*( ), *clock_nanosleep*( ), *ctime*( ), *mq_receive*( ), *mq_send*( ), *nanosleep*( ), *pthread_mutex_timedlock*( ), *sem_timedwait*( ), *time*( ), *timer_create*( ), *timer_getoverrun*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clock_nanosleep — high resolution sleep with specifiable clock

SYNOPSIS

```
#include <time.h>

int clock_nanosleep(clockid_t clock_id, int flags,
   const struct timespec *rqtp, struct timespec *rmtp);
```

DESCRIPTION

If the flag TIMER_ABSTIME is not set in the *flags* argument, the *clock_nanosleep*() function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. The clock used to measure the time shall be the clock specified by *clock_id*.

If the flag TIMER_ABSTIME is set in the *flags* argument, the *clock_nanosleep*() function shall cause the current thread to be suspended from execution until either the time value of the clock specified by *clock_id* reaches the absolute time specified by the *rqtp* argument, or a signal is delivered to the calling thread and its action is to invoke a signal-catching function, or the process is terminated. If, at the time of the call, the time value specified by *rqtp* is less than or equal to the time value of the specified clock, then *clock_nanosleep*() shall return immediately and the calling process shall not be suspended.

The suspension time caused by this function may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution, or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time for the relative *clock_nanosleep*() function (that is, with the TIMER_ABSTIME flag not set) shall not be less than the time interval specified by *rqtp*, as measured by the corresponding clock. The suspension for the absolute *clock_nanosleep*() function (that is, with the TIMER_ABSTIME flag set) shall be in effect at least until the value of the corresponding clock reaches the absolute time specified by *rqtp*, except for the case of being interrupted by a signal.

The use of the *clock_nanosleep*() function shall have no effect on the action or blockage of any signal.

The *clock_nanosleep*() function shall fail if the *clock_id* argument refers to the CPU-time clock of the calling thread. It is unspecified whether *clock_id* values of other CPU-time clocks are allowed.

RETURN VALUE

If the *clock_nanosleep*() function returns because the requested time has elapsed, its return value shall be zero.

If the *clock_nanosleep*() function returns because it has been interrupted by a signal, it shall return the corresponding error value. For the relative *clock_nanosleep*() function, if the *rmtp* argument is non-NULL, the **timespec** structure referenced by it shall be updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned. The absolute *clock_nanosleep*() function has no effect on the structure referenced by *rmtp*.

If *clock_nanosleep*() fails, it shall return the corresponding error value.

ERRORS

The *clock_nanosleep*() function shall fail if:

EINTR

The *clock_nanosleep*() function was interrupted by a signal.

EINVAL
The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million; or the TIMER_ABSTIME flag was specified in flags and the *rqtp* argument is outside the range for the clock specified by *clock_id*; or the *clock_id* argument does not specify a known clock, or specifies the CPU-time clock of the calling thread.

ENOTSUP
The *clock_id* argument specifies a clock for which *clock_nanosleep*() is not supported, such as a CPU-time clock.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Calling *clock_nanosleep*() with the value TIMER_ABSTIME not set in the *flags* argument and with a *clock_id* of CLOCK_REALTIME is equivalent to calling *nanosleep*() with the same *rqtp* and *rmtp* arguments.

RATIONALE

The *nanosleep*() function specifies that the system-wide clock CLOCK_REALTIME is used to measure the elapsed time for this time service. However, with the introduction of the monotonic clock CLOCK_MONOTONIC a new relative sleep function is needed to allow an application to take advantage of the special characteristics of this clock.

There are many applications in which a process needs to be suspended and then activated multiple times in a periodic way; for example, to poll the status of a non-interrupting device or to refresh a display device. For these cases, it is known that precise periodic activation cannot be achieved with a relative *sleep*() or *nanosleep*() function call. Suppose, for example, a periodic process that is activated at time $T0$, executes for a while, and then wants to suspend itself until time $T0+T$, the period being T. If this process wants to use the *nanosleep*() function, it must first call *clock_gettime*() to get the current time, then calculate the difference between the current time and $T0+T$ and, finally, call *nanosleep*() using the computed interval. However, the process could be preempted by a different process between the two function calls, and in this case the interval computed would be wrong; the process would wake up later than desired. This problem would not occur with the absolute *clock_nanosleep*() function, since only one function call would be necessary to suspend the process until the desired time. In other cases, however, a relative sleep is needed, and that is why both functionalities are required.

Although it is possible to implement periodic processes using the timers interface, this implementation would require the use of signals, and the reservation of some signal numbers. In this regard, the reasons for including an absolute version of the *clock_nanosleep*() function in POSIX.1-2008 are the same as for the inclusion of the relative *nanosleep*().

It is also possible to implement precise periodic processes using *pthread_cond_timedwait*(), in which an absolute timeout is specified that takes effect if the condition variable involved is never signaled. However, the use of this interface is unnatural, and involves performing other operations on mutexes and condition variables that imply an unnecessary overhead. Furthermore, *pthread_cond_timedwait*() is not available in implementations that do not support threads.

Although the interface of the relative and absolute versions of the new high resolution sleep service is the same *clock_nanosleep*() function, the *rmtp* argument is only used in the relative sleep. This argument is needed in the relative *clock_nanosleep*() function to reissue the function call if it is interrupted by a signal, but it is not needed in the absolute *clock_nanosleep*() function call; if the call is interrupted by a signal, the absolute *clock_nanosleep*() function can be invoked again with the same *rqtp* argument used in the interrupted call.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *nanosleep*( ), *pthread_cond_timedwait*( ), *sleep*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clock_settime — clock and timer functions

SYNOPSIS

#include <time.h>

int clock_settime(clockid_t *clock_id*, const struct timespec **tp*);

DESCRIPTION

Refer to *clock_getres*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

clog, clogf, clogl — complex natural logarithm functions

SYNOPSIS

```
#include <complex.h>

double complex clog(double complex z);
float complex clogf(float complex z);
long double complex clogl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex natural (base *e*) logarithm of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the complex natural logarithm value, in the range of a strip mathematically unbounded along the real axis and in the interval $[-i\pi, +i\pi]$ along the imaginary axis.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cexp( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

close — close a file descriptor

SYNOPSIS

```
#include <unistd.h>

int close(int fildes);
```

DESCRIPTION

The *close*() function shall deallocate the file descriptor indicated by *fildes*. To deallocate means to make the file descriptor available for return by subsequent calls to *open*() or other functions that allocate file descriptors. All outstanding record locks owned by the process on the file associated with the file descriptor shall be removed (that is, unlocked).

If *close*() is interrupted by a signal that is to be caught, it shall return −1 with *errno* set to **[EINTR]** and the state of *fildes* is unspecified. If an I/O error occurred while reading from or writing to the file system during *close*(), it may return −1 with *errno* set to **[EIO]**; if this error is returned, the state of *fildes* is unspecified.

When all file descriptors associated with a pipe or FIFO special file are closed, any data remaining in the pipe or FIFO shall be discarded.

When all file descriptors associated with an open file description have been closed, the open file description shall be freed.

If the link count of the file is 0, when all file descriptors associated with the file are closed, the space occupied by the file shall be freed and the file shall no longer be accessible.

If a STREAMS-based *fildes* is closed and the calling process was previously registered to receive a SIGPOLL signal for events associated with that STREAM, the calling process shall be unregistered for events associated with the STREAM. The last *close*() for a STREAM shall cause the STREAM associated with *fildes* to be dismantled. If O_NONBLOCK is not set and there have been no signals posted for the STREAM, and if there is data on the module's write queue, *close*() shall wait for an unspecified time (for each module and driver) for any output to drain before dismantling the STREAM. The time delay can be changed via an I_SETCLTIME *ioctl*() request. If the O_NONBLOCK flag is set, or if there are any pending signals, *close*() shall not wait for output to drain, and shall dismantle the STREAM immediately.

If the implementation supports STREAMS-based pipes, and *fildes* is associated with one end of a pipe, the last *close*() shall cause a hangup to occur on the other end of the pipe. In addition, if the other end of the pipe has been named by *fattach*(), then the last *close*() shall force the named end to be detached by *fdetach*(). If the named end has no open file descriptors associated with it and gets detached, the STREAM associated with that end shall also be dismantled.

If *fildes* refers to the master side of a pseudo-terminal, and this is the last close, a SIGHUP signal shall be sent to the controlling process, if any, for which the slave side of the pseudo-terminal is the controlling terminal. It is unspecified whether closing the master side of the pseudo-terminal flushes all queued input and output.

If *fildes* refers to the slave side of a STREAMS-based pseudo-terminal, a zero-length message may be sent to the master.

When there is an outstanding cancelable asynchronous I/O operation against *fildes* when *close*() is called, that I/O operation may be canceled. An I/O operation that is not canceled completes as if the *close*() operation had not yet occurred. All operations that are not canceled shall complete as if the *close*() blocked until the operations completed. The *close*() operation itself need not block awaiting such I/O completion. Whether any I/O operation is canceled, and which I/O operation may be canceled upon *close*(), is implementation-defined.

If a memory mapped file or a shared memory object remains referenced at the last close (that is, a process has it mapped), then the entire contents of the memory object shall persist until the memory object becomes unreferenced. If this is the last close of a memory mapped file or a shared memory object and the close results in the memory object becoming unreferenced, and the memory object has been unlinked, then the memory object shall be removed.

If *fildes* refers to a socket, *close*() shall cause the socket to be destroyed. If the socket is in connection-mode, and the SO_LINGER option is set for the socket with non-zero linger time, and the socket has untransmitted data, then *close*() shall block for up to the current linger interval until all data is transmitted.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *close*() function shall fail if:

EBADF
The *fildes* argument is not a open file descriptor.

EINTR
The *close*() function was interrupted by a signal.

The *close*() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.

The following sections are informative.

EXAMPLES

Reassigning a File Descriptor

The following example closes the file descriptor associated with standard output for the current process, reassigns standard output to a new file descriptor, and closes the original file descriptor to clean up. This example assumes that the file descriptor 0 (which is the descriptor for standard input) is not closed.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Incidentally, this is exactly what could be achieved using:

```
dup2(pfd, 1);
close(pfd);
```

Closing a File Descriptor

In the following example, *close*() is used to close a file descriptor after an unsuccessful attempt is made to associate that file descriptor with a stream.

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
```

```
FILE *fpfd;
...
if ((fpfd = fdopen (pfd, "w")) == NULL) {
   close(pfd);
   unlink(LOCKFILE);
   exit(1);
}
...
```

APPLICATION USAGE

An application that had used the *stdio* routine *fopen*() to open a file should use the corresponding *fclose*() routine rather than *close*(). Once a file is closed, the file descriptor no longer exists, since the integer corresponding to it no longer refers to a file.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *close*() on an arbitrary integer risks non-conforming behavior, and *close*() can only portably be used on file descriptor values that the application has obtained through explicit actions, as well as the three file descriptors corresponding to the standard file streams. In multi-threaded parent applications, the practice of calling *close*() in a loop after *fork*() and before an *exec* call in order to avoid a race condition of leaking an unintended file descriptor into a child process, is therefore unsafe, and the race should instead be combatted by opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

Usage of *close*() on file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO should immediately be followed by an operation to reopen these file descriptors. Unexpected behavior will result if any of these file descriptors is left in a closed state (for example, an **[EBADF]** error from *perror*()) or if an unrelated *open*() or similar call later in the application accidentally allocates a file to one of these well-known file descriptors. Furthermore, a *close*() followed by a reopen operation (e.g., *open*(), *dup*(), etc.) is not atomic; *dup2*() should be used to change standard file descriptors.

RATIONALE

The use of interruptible device close routines should be discouraged to avoid problems with the implicit closes of file descriptors by *exec* and *exit*(). This volume of POSIX.1-2017 only intends to permit such behavior by specifying the **[EINTR]** error condition.

Note that the requirement for *close*() on a socket to block for up to the current linger interval is not conditional on the O_NONBLOCK setting.

The standard developers rejected a proposal to add *closefrom*() to the standard. Because the standard permits implementations to use inherited file descriptors as a means of providing a conforming environment for the child process, it is not possible to standardize an interface that closes arbitrary file descriptors above a certain value while still guaranteeing a conforming environment.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6, *STREAMS*, *dup*( ), *exec*, *exit*( ), *fattach*( ), *fclose*( ), *fdetach*( ), *fopen*( ), *fork*( ), *ioctl*( ), *open*( ), *perror*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

closedir — close a directory stream

SYNOPSIS

```
#include <dirent.h>

int closedir(DIR *dirp);
```

DESCRIPTION

The *closedir*() function shall close the directory stream referred to by the argument *dirp*. Upon return, the value of *dirp* may no longer point to an accessible object of the type **DIR**. If a file descriptor is used to implement type **DIR**, that file descriptor shall be closed.

RETURN VALUE

Upon successful completion, *closedir*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *closedir*() function may fail if:

EBADF
The *dirp* argument does not refer to an open directory stream.

EINTR
The *closedir*() function was interrupted by a signal.

The following sections are informative.

EXAMPLES

Closing a Directory Stream

The following program fragment demonstrates how the *closedir*() function is used.

```
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
...
    }

    while ((dp = readdir (dir)) != NULL) {
...
    }

    closedir(dir);
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dirfd( ), *fdopendir*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

closelog, openlog, setlogmask, syslog — control system log

SYNOPSIS

```
#include <syslog.h>

void closelog(void);
void openlog(const char *ident, int logopt, int facility);
int setlogmask(int maskpri);
void syslog(int priority, const char *message, ... /* arguments */);
```

DESCRIPTION

The *syslog*() function shall send a message to an implementation-defined logging facility, which may log it in an implementation-defined system log, write it to the system console, forward it to a list of users, or forward it to the logging facility on another host over the network. The logged message shall include a message header and a message body. The message header contains at least a timestamp and a tag string.

The message body is generated from the *message* and following arguments in the same manner as if these were arguments to *printf*(), except that the additional conversion specification **%m** shall be recognized; it shall convert no arguments, shall cause the output of the error message string associated with the value of *errno* on entry to *syslog*(), and may be mixed with argument specifications of the "%*n*$" form. If a complete conversion specification with the **m** conversion specifier character is not just **%m**, the behavior is undefined. A trailing <newline> may be added if needed.

Values of the *priority* argument are formed by OR'ing together a severity-level value and an optional facility value. If no facility value is specified, the current default facility value is used.

Possible values of severity level include:

LOG_EMERG	A panic condition.
LOG_ALERT	A condition that should be corrected immediately, such as a corrupted system database.
LOG_CRIT	Critical conditions, such as hard device errors.
LOG_ERR	Errors.
LOG_WARNING	Warning messages.
LOG_NOTICE	Conditions that are not error conditions, but that may require special handling.
LOG_INFO	Informational messages.
LOG_DEBUG	Messages that contain information normally of use only when debugging a program.

The facility indicates the application or system component generating the message. Possible facility values include:

LOG_USER	Messages generated by arbitrary processes. This is the default facility identifier if none is specified.
LOG_LOCAL0	Reserved for local use.

LOG_LOCAL1
: Reserved for local use.

LOG_LOCAL2
: Reserved for local use.

LOG_LOCAL3
: Reserved for local use.

LOG_LOCAL4
: Reserved for local use.

LOG_LOCAL5
: Reserved for local use.

LOG_LOCAL6
: Reserved for local use.

LOG_LOCAL7
: Reserved for local use.

The *openlog*() function shall set process attributes that affect subsequent calls to *syslog*(). The *ident* argument is a string that is prepended to every message. The *logopt* argument indicates logging options. Values for *logopt* are constructed by a bitwise-inclusive OR of zero or more of the following:

LOG_PID Log the process ID with each message. This is useful for identifying specific processes.

LOG_CONS Write messages to the system console if they cannot be sent to the logging facility. The *syslog*() function ensures that the process does not acquire the console as a controlling terminal in the process of writing the message.

LOG_NDELAY
: Open the connection to the logging facility immediately. Normally the open is delayed until the first message is logged. This is useful for programs that need to manage the order in which file descriptors are allocated.

LOG_ODELAY
: Delay open until *syslog*() is called.

LOG_NOWAIT
: Do not wait for child processes that may have been created during the course of logging the message. This option should be used by processes that enable notification of child termination using SIGCHLD, since *syslog*() may otherwise block waiting for a child whose exit status has already been collected.

The *facility* argument encodes a default facility to be assigned to all messages that do not have an explicit facility already encoded. The initial default facility is LOG_USER.

The *openlog*() and *syslog*() functions may allocate a file descriptor. It is not necessary to call *openlog*() prior to calling *syslog*().

The *closelog*() function shall close any open file descriptors allocated by previous calls to *openlog*() or *syslog*().

The *setlogmask*() function shall set the log priority mask for the current process to *maskpri* and return the previous mask. If the *maskpri* argument is 0, the current log mask is not modified. Calls by the current process to *syslog*() with a priority not set in *maskpri* shall be rejected. The default log mask allows all priorities to be logged. A call to *openlog*() is not required prior to calling *setlogmask*().

Symbolic constants for use as values of the *logopt*, *facility*, *priority*, and *maskpri* arguments are defined in the <*syslog.h*> header.

RETURN VALUE

The *setlogmask*() function shall return the previous log priority mask. The *closelog*(), *openlog*(), and *syslog*() functions shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Using openlog( )

The following example causes subsequent calls to *syslog*() to log the process ID with each message, and to write messages to the system console if they cannot be sent to the logging facility.

```
#include <syslog.h>

char *ident = "Process demo";
int logopt = LOG_PID | LOG_CONS;
int facility = LOG_USER;
...
openlog(ident, logopt, facility);
```

Using setlogmask( )

The following example causes subsequent calls to *syslog*() to accept error messages, and to reject all other messages.

```
#include <syslog.h>

int result;
int mask = LOG_MASK (LOG_ERR);
...
result = setlogmask(mask);
```

Using syslog

The following example sends the message **"This**is**a**message" to the default logging facility, marking the message as an error message generated by random processes.

```
#include <syslog.h>

char *message = "This is a message";
int priority = LOG_ERR | LOG_USER;
...
syslog(priority, message);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( )

The Base Definitions volume of POSIX.1-2017, **<syslog.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

confstr — get configurable variables

SYNOPSIS

```
#include <unistd.h>

size_t confstr(int name, char *buf, size_t len);
```

DESCRIPTION

The *confstr*() function shall return configuration-defined string values. Its use and purpose are similar to *sysconf*(), but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support the following name values, defined in *<unistd.h>*. It may support others:

_CS_PATH
_CS_POSIX_V7_ILP32_OFF32_CFLAGS
_CS_POSIX_V7_ILP32_OFF32_LDFLAGS
_CS_POSIX_V7_ILP32_OFF32_LIBS
_CS_POSIX_V7_ILP32_OFFBIG_CFLAGS
_CS_POSIX_V7_ILP32_OFFBIG_LDFLAGS
_CS_POSIX_V7_ILP32_OFFBIG_LIBS
_CS_POSIX_V7_LP64_OFF64_CFLAGS
_CS_POSIX_V7_LP64_OFF64_LDFLAGS
_CS_POSIX_V7_LP64_OFF64_LIBS
_CS_POSIX_V7_LPBIG_OFFBIG_CFLAGS
_CS_POSIX_V7_LPBIG_OFFBIG_LDFLAGS
_CS_POSIX_V7_LPBIG_OFFBIG_LIBS
_CS_POSIX_V7_THREADS_CFLAGS
_CS_POSIX_V7_THREADS_LDFLAGS
_CS_POSIX_V7_WIDTH_RESTRICTED_ENVS
_CS_V7_ENV
_CS_POSIX_V6_ILP32_OFF32_CFLAGS
_CS_POSIX_V6_ILP32_OFF32_LDFLAGS
_CS_POSIX_V6_ILP32_OFF32_LIBS
_CS_POSIX_V6_ILP32_OFFBIG_CFLAGS
_CS_POSIX_V6_ILP32_OFFBIG_LDFLAGS
_CS_POSIX_V6_ILP32_OFFBIG_LIBS
_CS_POSIX_V6_LP64_OFF64_CFLAGS
_CS_POSIX_V6_LP64_OFF64_LDFLAGS
_CS_POSIX_V6_LP64_OFF64_LIBS
_CS_POSIX_V6_LPBIG_OFFBIG_CFLAGS
_CS_POSIX_V6_LPBIG_OFFBIG_LDFLAGS
_CS_POSIX_V6_LPBIG_OFFBIG_LIBS
_CS_POSIX_V6_WIDTH_RESTRICTED_ENVS
_CS_V6_ENV

If *len* is not 0, and if *name* has a configuration-defined value, *confstr*() shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr*() shall truncate the string to *len*−1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr*() with *len*.

If *len* is 0 and *buf* is a null pointer, then *confstr*() shall still return the integer value as defined below, but shall not return a string. If *len* is 0 but *buf* is not a null pointer, the result is unspecified.

After a call to:

```
    confstr(_CS_V7_ENV, buf, sizeof(buf))
```

the string stored in *buf* shall contain a <space>-separated list of the variable=value environment variable pairs an implementation requires as part of specifying a conforming environment, as described in the implementations' conformance documentation.

If the implementation supports the POSIX shell option, the string stored in *buf* after a call to:

```
    confstr(_CS_PATH, buf, sizeof(buf))
```

can be used as a value of the *PATH* environment variable that accesses all of the standard utilities of POSIX.1-2008, that are provided in a manner accessible via the *exec* family of functions, if the return value is less than or equal to *sizeof*(*buf*).

RETURN VALUE

If *name* has a configuration-defined value, *confstr*() shall return the size of buffer that would be needed to hold the entire configuration-defined value including the terminating null. If this return value is greater than *len*, the string returned in *buf* is truncated.

If *name* is invalid, *confstr*() shall return 0 and set *errno* to indicate the error.

If *name* does not have a configuration-defined value, *confstr*() shall return 0 and leave *errno* unchanged.

ERRORS

The *confstr*() function shall fail if:

EINVAL
The value of the *name* argument is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

An application can distinguish between an invalid *name* parameter value and one that corresponds to a configurable variable that has no configuration-defined value by checking if *errno* is modified. This mirrors the behavior of *sysconf*().

The original need for this function was to provide a way of finding the configuration-defined default value for the environment variable *PATH*. Since *PATH* can be modified by the user to include directories that could contain utilities replacing the standard utilities in the Shell and Utilities volume of POSIX.1-2017, applications need a way to determine the system-supplied *PATH* environment variable value that contains the correct search path for the standard utilities.

An application could use:

```
    confstr(name, (char *)NULL, (size_t)0)
```

to find out how big a buffer is needed for the string value; use *malloc*() to allocate a buffer to hold the string; and call *confstr*() again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most answers (perhaps 512 or 1 024 bytes), but then use *malloc*() to allocate a larger buffer if it finds that this is too small.

RATIONALE

Application developers can normally determine any configuration variable by means of reading from the stream opened by a call to:

```
    popen("command -p getconf variable", "r");
```

The *confstr*() function with a *name* argument of _CS_PATH returns a string that can be used as a *PATH* environment variable setting that will reference the standard shell and utilities as described in the Shell and Utilities volume of POSIX.1-2017.

The *confstr*() function copies the returned string into a buffer supplied by the application instead of returning a pointer to a string. This allows a cleaner function in some implementations (such as those with lightweight threads) and resolves questions about when the application must copy the string returned.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fpathconf*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

The Shell and Utilities volume of POSIX.1-2017, *c99*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

conj, conjf, conjl — complex conjugate functions

SYNOPSIS

```
#include <complex.h>

double complex conj(double complex z);
float complex conjf(float complex z);
long double complex conjl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex conjugate of *z*, by reversing the sign of its imaginary part.

RETURN VALUE

These functions return the complex conjugate value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *cproj*( ), *creal*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

connect — connect a socket

SYNOPSIS

```
#include <sys/socket.h>

int connect(int socket, const struct sockaddr *address,
    socklen_t address_len);
```

DESCRIPTION

The *connect*() function shall attempt to make a connection on a connection-mode socket or to set or reset the peer address of a connectionless-mode socket. The function takes the following arguments:

socket	Specifies the file descriptor associated with the socket.
address	Points to a **sockaddr** structure containing the peer address. The length and format of the address depend on the address family of the socket.
address_len	Specifies the length of the **sockaddr** structure pointed to by the *address* argument.

If the socket has not already been bound to a local address, *connect*() shall bind it to an address which, unless the socket's address family is AF_UNIX, is an unused local address.

If the initiating socket is not connection-mode, then *connect*() shall set the socket's peer address, and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent *send*() functions, and limits the remote sender for subsequent *recv*() functions. If the *sa_family* member of *address* is AF_UNSPEC, the socket's peer address shall be reset. Note that despite no connection being made, the term "connected" is used to describe a connectionless-mode socket for which a peer address has been set.

If the initiating socket is connection-mode, then *connect*() shall attempt to establish a connection to the address specified by the *address* argument. If the connection cannot be established immediately and O_NONBLOCK is not set for the file descriptor for the socket, *connect*() shall block for up to an unspecified timeout interval until the connection is established. If the timeout interval expires before the connection is established, *connect*() shall fail and the connection attempt shall be aborted. If *connect*() is interrupted by a signal that is caught while blocked waiting to establish a connection, *connect*() shall fail and set *errno* to **[EINTR]**, but the connection request shall not be aborted, and the connection shall be established asynchronously.

If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, *connect*() shall fail and set *errno* to **[EINPROGRESS]**, but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to *connect*() for the same socket, before the connection is established, shall fail and set *errno* to **[EALREADY]**.

When the connection has been established asynchronously, *pselect*(), *select*(), and *poll*() shall indicate that the file descriptor for the socket is ready for writing.

The socket in use may require the process to have appropriate privileges to use the *connect*() function.

RETURN VALUE

Upon successful completion, *connect*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *connect*() function shall fail if:

EADDRNOTAVAIL
The specified address is not available from the local machine.

EAFNOSUPPORT
The specified address is not a valid address for the address family of the specified socket.

EALREADY
A connection request is already in progress for the specified socket.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNREFUSED
The target address was not listening for connections or refused the connection request.

EINPROGRESS
O_NONBLOCK is set for the file descriptor for the socket and the connection cannot be immediately established; the connection shall be established asynchronously.

EINTR
The attempt to establish a connection was interrupted by delivery of a signal that was caught; the connection shall be established asynchronously.

EISCONN
The specified socket is connection-mode and is already connected.

ENETUNREACH
No route to the network is present.

ENOTSOCK
The *socket* argument does not refer to a socket.

EPROTOTYPE
The specified address has a different type than the socket bound to the specified peer address.

ETIMEDOUT
The attempt to connect timed out before a connection was made.

If the address family of the socket is AF_UNIX, then *connect*() shall fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the pathname in *address*.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the pathname does not name an existing file or the pathname is an empty string.

ENOTDIR
A component of the path prefix of the pathname in *address* names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in *address* contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *connect*() function may fail if:

EACCES
Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

EADDRINUSE
Attempt to establish a connection that uses addresses that are already in use.

ECONNRESET
Remote host reset the connection request.

EHOSTUNREACH
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

EINVAL
The *address_len* argument is not a valid length for the address family; or invalid address family in the **sockaddr** structure.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in *address*.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENETDOWN
The local network interface used to reach the destination is down.

ENOBUFS
No buffer space is available.

EOPNOTSUPP
The socket is listening and cannot be connected.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If *connect*() fails, the state of the socket is unspecified. Conforming applications should close the file descriptor and create a new socket before attempting to reconnect.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *close*( ), *getsockname*( ), *poll*( ), *pselect*( ), *send*( ), *shutdown*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

copysign, copysignf, copysignl — number manipulation function

SYNOPSIS

#include <math.h>

double copysign(double *x*, double *y*);
float copysignf(float *x*, float *y*);
long double copysignl(long double *x*, long double *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall produce a value with the magnitude of *x* and the sign of *y*. On implementations that represent a signed zero but do not treat negative zero consistently in arithmetic operations, these functions regard the sign of zero as positive.

RETURN VALUE

Upon successful completion, these functions shall return a value with the magnitude of *x* and the sign of *y*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

signbit( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cos, cosf, cosl — cosine function

SYNOPSIS

```
#include <math.h>

double cos(double x);
float cosf(float x);
long double cosl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the cosine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the cosine of *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, the value 1.0 shall be returned.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error
: The *x* argument is ±Inf.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Taking the Cosine of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45 * M_PI / 180;
double result;
...
result = cos(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near an odd multiple of $\pi/2$ or is far from 0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acos( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sin*( ), *tan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cosh, coshf, coshl — hyperbolic cosine functions

SYNOPSIS

```
#include <math.h>

double cosh(double x);
float coshf(float x);
long double coshl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the hyperbolic cosine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic cosine of *x*.

If the correct value would cause overflow, a range error shall occur and *cosh*(), *coshf*(), and *coshl*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, the value 1.0 shall be returned.

If *x* is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Range Error The result would cause an overflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

acosh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sinh*( ), *tanh*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for*

Mathematical Functions, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cosl — cosine function

SYNOPSIS

```
#include <math.h>

long double cosl(long double x);
```

DESCRIPTION

Refer to *cos*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cpow, cpowf, cpowl — complex power functions

SYNOPSIS

```
#include <complex.h>

double complex cpow(double complex x, double complex y);
float complex cpowf(float complex x, float complex y);
long double complex cpowl(long double complex x,
    long double complex y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex power function x^y, with a branch cut for the first parameter along the negative real axis.

RETURN VALUE

These functions shall return the complex power function value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cabs( ), *csqrt*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

cproj, cprojf, cprojl — complex projection functions

SYNOPSIS

```
#include <complex.h>

double complex cproj(double complex z);
float complex cprojf(float complex z);
long double complex cprojl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute a projection of *z* onto the Riemann sphere: *z* projects to *z*, except that all complex infinities (even those with one infinite part and one NaN part) project to positive infinity on the real axis. If *z* has an infinite part, then *cproj*(*z*) shall be equivalent to:

```
INFINITY + I * copysign(0.0, cimag(z))
```

RETURN VALUE

These functions shall return the value of the projection onto the Riemann sphere.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Two topologies are commonly used in complex mathematics: the complex plane with its continuum of infinities, and the Riemann sphere with its single infinity. The complex plane is better suited for transcendental functions, the Riemann sphere for algebraic functions. The complex types with their multiplicity of infinities provide a useful (though imperfect) model for the complex plane. The *cproj*() function helps model the Riemann sphere by mapping all infinities to one, and should be used just before any operation, especially comparisons, that might give spurious results for any of the other infinities. Note that a complex value with one infinite part and one NaN part is regarded as an infinity, not a NaN, because if one part is infinite, the complex value is infinite independent of the value of the other part. For the same reason, *cabs*() returns an infinity if its argument has an infinite part and a NaN part.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *conj*( ), *creal*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

creal, crealf, creall — complex real functions

SYNOPSIS

#include <complex.h>

double creal(double complex *z*);
float crealf(float complex *z*);
long double creall(long double complex *z*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the real part of *z*.

RETURN VALUE

These functions shall return the real part value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

For a variable *z* of type **complex**:

```
z == creal(z) + cimag(z)*I
```

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

carg( ), *cimag*( ), *conj*( ), *cproj*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

creat — create a new file or rewrite an existing one

SYNOPSIS

```
#include <sys/stat.h>
#include <fcntl.h>

int creat(const char *path, mode_t mode);
```

DESCRIPTION

The *creat*() function shall behave as if it is implemented as follows:

```
int creat(const char *path, mode_t mode)
{
    return open(path, O_WRONLY|O_CREAT|O_TRUNC, mode);
}
```

RETURN VALUE

Refer to *open*( ).

ERRORS

Refer to *open*( ).

The following sections are informative.

EXAMPLES

Creating a File

The following example creates the file **/tmp/file** with read and write permissions for the file owner and read permission for group and others. The resulting file descriptor is assigned to the *fd* variable.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *pathname = "/tmp/file";
...
fd = creat(pathname, mode);
...
```

APPLICATION USAGE

None.

RATIONALE

The *creat*() function is redundant. Its services are also provided by the *open*() function. It has been included primarily for historical purposes since many existing applications depend on it. It is best considered a part of the C binding rather than a function that should be provided in other languages.

FUTURE DIRECTIONS

None.

SEE ALSO

mknod( ), *open*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

crypt — string encoding function (**CRYPT**)

SYNOPSIS

#include <unistd.h>

char *crypt(const char **key*, const char **salt*);

DESCRIPTION

The *crypt*() function is a string encoding function. The algorithm is implementation-defined.

The *key* argument points to a string to be encoded. The *salt* argument shall be a string of at least two bytes in length not including the null character chosen from the set:

```
a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
0 1 2 3 4 5 6 7 8 9 . /
```

The first two bytes of this string may be used to perturb the encoding algorithm.

The return value of *crypt*() points to static data that is overwritten by each call.

The *crypt*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *crypt*() shall return a pointer to the encoded string. The first two bytes of the returned value shall be those of the *salt* argument. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *crypt*() function shall fail if:

ENOSYS
The functionality is not supported on this implementation.

The following sections are informative.

EXAMPLES

Encoding Passwords

The following example finds a user database entry matching a particular user name and changes the current password to a new password. The *crypt*() function generates an encoded version of each password. The first call to *crypt*() produces an encoded version of the old password; that encoded password is then compared to the password stored in the user database. The second call to *crypt*() encodes the new password before it is stored.

The *putpwent*() function, used in the following example, is not part of POSIX.1-2008.

```
#include <unistd.h>
#include <pwd.h>
#include <string.h>
#include <stdio.h>
...
int valid_change;
int pfd;  /* Integer for file descriptor returned by open(). */
FILE *fpfd;  /* File pointer for use in putpwent(). */
struct passwd *p;
```

```
char user[100];
char oldpasswd[100];
char newpasswd[100];
char savepasswd[100];
...
valid_change = 0;
while ((p = getpwent()) != NULL) {
  /* Change entry if found. */
  if (strcmp(p->pw_name, user) == 0) {
    if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
      strcpy(savepasswd, crypt(newpasswd, user));
      p->pw_passwd = savepasswd;
      valid_change = 1;
    }
    else {
      fprintf(stderr, "Old password is not valid\n");
    }
  }
  /* Put passwd entry into ptmp. */
  putpwent(p, fpfd);
}
```

APPLICATION USAGE

The values returned by this function need not be portable among XSI-conformant systems.

Several implementations offer extensions via characters outside of the set specified for the *salt* argument for specifying alternative algorithms; while not portable, these extensions may offer better security. The use of *crypt*() for anything other than password hashing is not recommended.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

encrypt( ), *setkey*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

csin, csinf, csinl — complex sine functions

SYNOPSIS

```
#include <complex.h>

double complex csin(double complex z);
float complex csinf(float complex z);
long double complex csinl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex sine of *z*.

RETURN VALUE

These functions shall return the complex sine value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

casin( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

csinh, csinhf, csinhl — complex hyperbolic sine functions

SYNOPSIS

```
#include <complex.h>

double complex csinh(double complex z);
float complex csinhf(float complex z);
long double complex csinhl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex hyperbolic sine of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic sine value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

casinh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

csinl — complex sine functions

SYNOPSIS

```
#include <complex.h>

long double complex csinl(long double complex z);
```

DESCRIPTION

Refer to *csin*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

csqrt, csqrtf, csqrtl — complex square root functions

SYNOPSIS

```
#include <complex.h>

double complex csqrt(double complex z);
float complex csqrtf(float complex z);
long double complex csqrtl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex square root of *z*, with a branch cut along the negative real axis.

RETURN VALUE

These functions shall return the complex square root value, in the range of the right half-plane (including the imaginary axis).

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

cabs( ), *cpow*( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ctan, ctanf, ctanl — complex tangent functions

SYNOPSIS

```
#include <complex.h>

double complex ctan(double complex z);
float complex ctanf(float complex z);
long double complex ctanl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex tangent of *z*.

RETURN VALUE

These functions shall return the complex tangent value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catan( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ctanh, ctanhf, ctanhl — complex hyperbolic tangent functions

SYNOPSIS

```
#include <complex.h>

double complex ctanh(double complex z);
float complex ctanhf(float complex z);
long double complex ctanhl(long double complex z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complex hyperbolic tangent of *z*.

RETURN VALUE

These functions shall return the complex hyperbolic tangent value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

catanh( )

The Base Definitions volume of POSIX.1-2017, **<complex.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ctanl — complex tangent functions

SYNOPSIS

#include <complex.h>

long double complex ctanl(long double complex *z*);

DESCRIPTION

Refer to *ctan*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ctermid — generate a pathname for the controlling terminal

SYNOPSIS

#include <stdio.h>

char *ctermid(char **s*);

DESCRIPTION

The *ctermid*() function shall generate a string that, when used as a pathname, refers to the current controlling terminal for the current process. If *ctermid*() returns a pathname, access to the file is not guaranteed.

The *ctermid*() function need not be thread-safe if called with a NULL parameter.

RETURN VALUE

If *s* is a null pointer, the string shall be generated in an area that may be static, the address of which shall be returned. The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ctermid*(). The returned pointer might also be invalidated if the calling thread is terminated. If *s* is not a null pointer, *s* is assumed to point to a character array of at least L_ctermid bytes; the string is placed in this array and the value of *s* shall be returned. The symbolic constant L_ctermid is defined in *<stdio.h>*, and shall have a value greater than 0.

The *ctermid*() function shall return an empty string if the pathname that would refer to the controlling terminal cannot be determined, or if the function is unsuccessful.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Determining the Controlling Terminal for the Current Process

The following example returns a pointer to a string that identifies the controlling terminal for the current process. The pathname for the terminal is stored in the array pointed to by the *ptr* argument, which has a size of L_ctermid bytes, as indicated by the *term* argument.

```
#include <stdio.h>
...
char term[L_ctermid];
char *ptr;

ptr = ctermid(term);
```

APPLICATION USAGE

The difference between *ctermid*() and *ttyname*() is that *ttyname*() must be handed a file descriptor and return a path of the terminal associated with that file descriptor, while *ctermid*() returns a string (such as **"/dev/tty"**) that refers to the current controlling terminal if used as a pathname.

RATIONALE

L_ctermid must be defined appropriately for a given implementation and must be greater than zero so that array declarations using it are accepted by the compiler. The value includes the terminating null byte.

Conforming applications that use multiple threads cannot call *ctermid*() with NULL as the parameter. If *s* is not NULL, the *ctermid*() function generates a string that, when used as a pathname, refers to the current controlling terminal for the current process. If *s* is NULL, the return value of *ctermid*() is undefined.

There is no additional burden on the programmer—changing to use a hypothetical thread-safe version of *ctermid*() along with allocating a buffer is more of a burden than merely allocating a buffer. Application code should not assume that the returned string is short, as some implementations have more than two pathname components before reaching a logical device name.

FUTURE DIRECTIONS

None.

SEE ALSO

ttyname( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ctime, ctime_r — convert a time value to a date and time string

SYNOPSIS

```
#include <time.h>

char *ctime(const time_t *clock);
char *ctime_r(const time_t *clock, char *buf);
```

DESCRIPTION

For *ctime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ctime*() function shall convert the time pointed to by *clock*, representing time in seconds since the Epoch, to local time in the form of a string. It shall be equivalent to:

```
asctime(localtime(clock))
```

The *asctime*(), *ctime*(), *gmtime*(), and *localtime*() functions shall return values in one of two static objects: a broken-down time structure and an array of **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *ctime*() function need not be thread-safe.

The *ctime_r*() function shall convert the calendar time pointed to by *clock* to local time in exactly the same form as *ctime*() and put the string into the array pointed to by *buf* (which shall be at least 26 bytes in size) and return *buf*.

Unlike *ctime*(), the *ctime_r*() function is not required to set *tzname*. If *ctime_r*() sets *tzname*, it shall also set *daylight* and *timezone*. If *ctime_r*() does not set *tzname*, it shall not set *daylight* and shall not set *timezone*.

RETURN VALUE

The *ctime*() function shall return the pointer returned by *asctime*() with that broken-down time as an argument.

Upon successful completion, *ctime_r*() shall return a pointer to the string pointed to by *buf*. When an error is encountered, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are included only for compatibility with older implementations. They have undefined behavior if the resulting string would be too long, so the use of these functions should be discouraged. On implementations that do not detect output string length overflow, it is possible to overflow the output buffers in such a way as to cause applications to fail, or possible system security violations. Also, these functions do not support localized date and time formats. To avoid these problems, applications should use *strftime*() to generate strings from broken-down times.

Values for the broken-down time structure can be obtained by calling *gmtime*() or *localtime*().

The *ctime_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Attempts to use *ctime*() or *ctime_r*() for times before the Epoch or for times beyond the year 9999 produce undefined results. Refer to *asctime*( ).

RATIONALE

The standard developers decided to mark the *ctime*() and *ctime_r*() functions obsolescent even though they are in the ISO C standard due to the possibility of buffer overflow. The ISO C standard also provides the *strftime*() function which can be used to avoid these problems.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

asctime( ), *clock*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

daylight — daylight savings time flag

SYNOPSIS

```
#include <time.h>

extern int daylight;
```

DESCRIPTION

Refer to *tzset*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dbm_clearerr, dbm_close, dbm_delete, dbm_error, dbm_fetch, dbm_firstkey, dbm_nextkey, dbm_open, dbm_store — database functions

SYNOPSIS

```
#include <ndbm.h>

int dbm_clearerr(DBM *db);
void dbm_close(DBM *db);
int dbm_delete(DBM *db, datum key);
int dbm_error(DBM *db);
datum dbm_fetch(DBM *db, datum key);
datum dbm_firstkey(DBM *db);
datum dbm_nextkey(DBM *db);
DBM *dbm_open(const char *file, int open_flags, mode_t file_mode);
int dbm_store(DBM *db, datum key, datum content, int store_mode);
```

DESCRIPTION

These functions create, access, and modify a database.

A **datum** consists of at least two members, *dptr* and *dsize*. The *dptr* member points to an object that is *dsize* bytes in length. Arbitrary binary data, as well as character strings, may be stored in the object pointed to by *dptr*.

A database shall be stored in one or two files. When one file is used, the name of the database file shall be formed by appending the suffix **.db** to the *file* argument given to *dbm_open*(). When two files are used, the names of the database files shall be formed by appending the suffixes **.dir** and **.pag** respectively to the *file* argument.

The *dbm_open*() function shall open a database. The *file* argument to the function is the pathname of the database. The *open_flags* argument has the same meaning as the *flags* argument of *open*() except that a database opened for write-only access opens the files for read and write access and the behavior of the O_APPEND flag is unspecified. The *file_mode* argument has the same meaning as the third argument of *open*().

The *dbm_open*() function need not accept pathnames longer than {PATH_MAX}−4 bytes (including the terminating null), or pathnames with a last component longer than {NAME_MAX}−4 bytes (excluding the terminating null).

The *dbm_close*() function shall close a database. The application shall ensure that argument *db* is a pointer to a **dbm** structure that has been returned from a call to *dbm_open*().

These database functions shall support an internal block size large enough to support key/content pairs of at least 1 023 bytes.

The *dbm_fetch*() function shall read a record from a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*(). The argument *key* is a **datum** that has been initialized by the application to the value of the key that matches the key of the record the program is fetching.

The *dbm_store*() function shall write a record to a database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*(). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies (for subsequent reading, writing, or deleting) the record the application is writing. The argument *content* is a **datum** that has been initialized by the application to the value of the record the program is writing. The argument *store_mode* controls whether *dbm_store*() replaces any pre-existing record that has the same key that is specified by the *key* argument. The application shall set *store_mode* to either DBM_INSERT or DBM_REPLACE. If the database contains a record that matches the *key* argument and *store_mode* is DBM_REPLACE, the existing record shall be replaced with the new record. If the database contains a record that matches the *key* argument and *store_mode* is DBM_INSERT, the existing record shall be left unchanged and the new record ignored. If the database does not contain a record that matches the *key* argument and *store_mode* is either DBM_INSERT or DBM_REPLACE, the new record shall be inserted in the database.

If the sum of a key/content pair exceeds the internal block size, the result is unspecified. Moreover, the application shall ensure that all key/content pairs that hash together fit on a single block. The *dbm_store*() function shall return an error in the event that a disk block fills with inseparable data.

The *dbm_delete*() function shall delete a record and its key from the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*(). The argument *key* is a **datum** that has been initialized by the application to the value of the key that identifies the record the program is deleting.

The *dbm_firstkey*() function shall return the first key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*().

The *dbm_nextkey*() function shall return the next key in the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*(). The application shall ensure that the *dbm_firstkey*() function is called before calling *dbm_nextkey*(). Subsequent calls to *dbm_nextkey*() return the next key until all of the keys in the database have been returned.

The *dbm_error*() function shall return the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*().

The *dbm_clearerr*() function shall clear the error condition of the database. The argument *db* is a pointer to a database structure that has been returned from a call to *dbm_open*().

The *dptr* pointers returned by these functions may point into static storage that may be changed by subsequent calls.

These functions need not be thread-safe.

RETURN VALUE

The *dbm_store*() and *dbm_delete*() functions shall return 0 when they succeed and a negative value when they fail.

The *dbm_store*() function shall return 1 if it is called with a *flags* value of DBM_INSERT and the function finds an existing record with the same key.

The *dbm_error*() function shall return 0 if the error condition is not set and return a non-zero value if the error condition is set.

The return value of *dbm_clearerr*() is unspecified.

The *dbm_firstkey*() and *dbm_nextkey*() functions shall return a key **datum**. When the end of the database is reached, the *dptr* member of the key is a null pointer. If an error is detected, the *dptr* member of the key shall be a null pointer and the error condition of the database shall be set.

The *dbm_fetch*() function shall return a content **datum**. If no record in the database matches the key or if an error condition has been detected in the database, the *dptr* member of the content shall be a null pointer.

The *dbm_open*() function shall return a pointer to a database structure. If an error is detected during the operation, *dbm_open*() shall return a (**DBM ***)0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The following code can be used to traverse the database:

```
for(key = dbm_firstkey(db); key.dptr != NULL; key = dbm_nextkey(db))
```

The *dbm_** functions provided in this library should not be confused in any way with those of a general-purpose database management system. These functions do not provide for multiple search keys per entry, they do not protect against multi-user access (in other words they do not lock records or files), and they do not provide the many other useful database functions that are found in more robust database management systems. Creating and updating databases by use of these functions is relatively slow because of data copies that occur upon hash collisions. These functions are useful for applications requiring fast lookup of relatively static information that is to be indexed by a single key.

Note that a strictly conforming application is extremely limited by these functions: since there is no way to determine that the keys in use do not all hash to the same value (although that would be rare), a strictly conforming application cannot be guaranteed that it can store more than one block's worth of data in the database. As long as a key collision does not occur, additional data may be stored, but because there is no way to determine whether an error is due to a key collision or some other error condition (*dbm_error*() being effectively a Boolean), once an error is detected, the application is effectively limited to guessing what the error might be if it wishes to continue using these functions.

The *dbm_delete*() function need not physically reclaim file space, although it does make it available for reuse by the database.

After calling *dbm_store*() or *dbm_delete*() during a pass through the keys by *dbm_firstkey*() and *dbm_nextkey*(), the application should reset the database by calling *dbm_firstkey*() before again calling *dbm_nextkey*(). The contents of these files are unspecified and may not be portable.

Applications should take care that database pathname arguments specified to *dbm_open*() are not prefixes of unrelated files. This might be done, for example, by placing databases in a separate directory.

Since some implementations use three characters for a suffix and others use four characters for a suffix, applications should ensure that the maximum portable pathname length passed to *dbm_open*() is no greater than {PATH_MAX}−4 bytes, with the last component of the pathname no greater than {NAME_MAX}−4 bytes.

RATIONALE

Previously the standard required the database to be stored in two files, one file being a directory containing a bitmap of keys and having **.dir** as its suffix. The second file containing all data and having **.pag** as its suffix. This has been changed not to specify the use of the files and to allow newer implementations of the Berkeley DB interface using a single file that have evolved while remaining compatible with the application programming interface. The standard developers considered removing the specific suffixes altogether but decided to retain them so as not to pollute the application file name space more than necessary and to allow for portable backups of the database.

FUTURE DIRECTIONS

None.

SEE ALSO

open( )

The Base Definitions volume of POSIX.1-2017, **<ndbm.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

difftime — compute the difference between two calendar time values

SYNOPSIS

#include <time.h>

double difftime(time_t *time1*, time_t *time0*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *difftime*() function shall compute the difference between two calendar times (as returned by *time*()): *time*1− *time*0.

RETURN VALUE

The *difftime*() function shall return the difference expressed in seconds as a type **double**.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dirfd — extract the file descriptor used by a DIR stream

SYNOPSIS

```
#include <dirent.h>

int dirfd(DIR *dirp);
```

DESCRIPTION

The *dirfd*() function shall return a file descriptor referring to the same directory as the *dirp* argument. This file descriptor shall be closed by a call to *closedir*(). If any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*(), *readdir*(), *readdir_r*(), *rewinddir*(), or *seekdir*(), the behavior is undefined.

RETURN VALUE

Upon successful completion, the *dirfd*() function shall return an integer which contains a file descriptor for the stream pointed to by *dirp*. Otherwise, it shall return −1 and shall set *errno* to indicate the error.

ERRORS

The *dirfd*() function may fail if:

EINVAL
The *dirp* argument does not refer to a valid directory stream.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *dirfd*() function is intended to be a mechanism by which an application may obtain a file descriptor to use for the *fchdir*() function.

RATIONALE

This interface was introduced because the Base Definitions volume of POSIX.1-2017 does not make public the **DIR** data structure. Applications tend to use the *fchdir*() function on the file descriptor returned by this interface, and this has proven useful for security reasons; in particular, it is a better technique than others where directory names might change.

The description uses the term "a file descriptor" rather than "the file descriptor". The implication intended is that an implementation that does not use an *fd* for *opendir*() could still *open*() the directory to implement the *dirfd*() function. Such a descriptor must be closed later during a call to *closedir*().

If it is necessary to allocate an *fd* to be returned by *dirfd*(), it should be done at the time of a call to *opendir*().

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *fchdir*( ), *fdopendir*( ), *fileno*( ), *open*( ), *readdir*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dirname — report the parent directory name of a file pathname

SYNOPSIS

#include <libgen.h>

char *dirname(char **path*);

DESCRIPTION

The *dirname*() function shall take a pointer to a character string that contains a pathname, and return a pointer to a string that is a pathname of the parent directory of that file. The *dirname*() function shall not perform pathname resolution; the result shall not be affected by whether or not *path* exists or by its file type. Trailing **'/'** characters in the path that are not also leading **'/'** characters shall not be counted as part of the path.

If *path* does not contain a **'/'**, then *dirname*() shall return a pointer to the string **"."**. If *path* is a null pointer or points to an empty string, *dirname*() shall return a pointer to the string **"."**.

The *dirname*() function may modify the string pointed to by *path*, and may return a pointer to static storage that may then be overwritten by a subsequent call to *dirname*().

The *dirname*() function need not be thread-safe.

RETURN VALUE

The *dirname*() function shall return a pointer to a string as described above.

The *dirname*() function may modify the string pointed to by *path*, and may return a pointer to internal storage. The returned pointer might be invalidated or the storage might be overwritten by a subsequent call to *dirname*(). The returned pointer might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following code fragment reads a pathname, changes the current working directory to the parent directory, and opens the file.

```
char *path = NULL, *pathcopy;
size_t buflen = 0;
ssize_t linelen = 0;
int fd;

linelen = getline(&path, &buflen, stdin);

path[linelen-1] = 0;
pathcopy = strdup(path);
if (chdir(dirname(pathcopy)) < 0) {
    ...
}
if ((fd = open(basename(path), O_RDONLY)) >= 0) {
    ...
    close (fd);
}
...
free (pathcopy);
```

```
    free (path);
```

The EXAMPLES section of the *basename*() function (see *basename*( )) includes a table showing examples of the results of processing several sample pathnames by the *basename*() and *dirname*() functions and by the *basename* and *dirname* utilities.

APPLICATION USAGE

The *dirname*() and *basename*() functions together yield a complete pathname. The expression *dirname*(*path*) obtains the pathname of the directory where *basename*(*path*) is found.

Since the meaning of the leading **"//"** is implementation-defined, *dirname*("**//foo**) may return either **"//"** or **'/'** (but nothing else).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

basename( )

The Base Definitions volume of POSIX.1-2017, **<libgen.h>**

The Shell and Utilities volume of POSIX.1-2017, *basename*, *dirname*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

div — compute the quotient and remainder of an integer division

SYNOPSIS

#include <stdlib.h>

div_t div(int *numer*, int *denom*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *div*() function shall compute the quotient and remainder of the division of the numerator *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the integer of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, *quot*denom+rem* shall equal *numer*.

RETURN VALUE

The *div*() function shall return a structure of type **div_t**, comprising both the quotient and the remainder. The structure includes the following members, in any order:

```
int  quot;  /* quotient */
int  rem;   /* remainder */
```

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ldiv( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dlclose — close a symbol table handle

SYNOPSIS

#include <dlfcn.h>

int dlclose(void **handle*);

DESCRIPTION

The *dlclose*() function shall inform the system that the symbol table handle specified by *handle* is no longer needed by the application.

An application writer may use *dlclose*() to make a statement of intent on the part of the process, but this statement does not create any requirement upon the implementation. When the symbol table handle is closed, the implementation may unload the executable object files that were loaded by *dlopen*() when the symbol table handle was opened and those that were loaded by *dlsym*() when using the symbol table handle identified by *handle*.

Once a symbol table handle has been closed, an application should assume that any symbols (function identifiers and data object identifiers) made visible using *handle*, are no longer available to the process.

Although a *dlclose*() operation is not required to remove any functions or data objects from the address space, neither is an implementation prohibited from doing so. The only restriction on such a removal is that no function nor data object shall be removed to which references have been relocated, until or unless all such references are removed. For instance, an executable object file that had been loaded with a *dlopen*() operation specifying the RTLD_GLOBAL flag might provide a target for dynamic relocations performed in the processing of other relocatable objects—in such environments, an application may assume that no relocation, once made, shall be undone or remade unless the executable object file containing the relocated object has itself been removed.

RETURN VALUE

If the referenced symbol table handle was successfully closed, *dlclose*() shall return 0. If *handle* does not refer to an open symbol table handle or if the symbol table handle could not be closed, *dlclose*() shall return a non-zero value. More detailed diagnostic information shall be available through *dlerror*().

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example illustrates use of *dlopen*() and *dlclose*():

```
#include <dlfcn.h>
int eret;
void *mylib;
...
/* Open a dynamic library and then close it ... */
mylib = dlopen("mylib.so", RTLD_LOCAL | RTLD_LAZY);
...
eret = dlclose(mylib);
...
```

APPLICATION USAGE

A conforming application should employ a symbol table handle returned from a *dlopen*() invocation only within a given scope bracketed by a *dlopen*() operation and the corresponding *dlclose*() operation.

Implementations are free to use reference counting or other techniques such that multiple calls to *dlopen*() referencing the same executable object file may return a pointer to the same data object as the symbol table handle.

Implementations are also free to re-use a handle. For these reasons, the value of a handle must be treated as an opaque data type by the application, used only in calls to *dlsym*() and *dlclose*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlerror( ), *dlopen*( ), *dlsym*( )

The Base Definitions volume of POSIX.1-2017, **<dlfcn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dlerror — get diagnostic information

SYNOPSIS

```
#include <dlfcn.h>

char *dlerror(void);
```

DESCRIPTION

The *dlerror*() function shall return a null-terminated character string (with no trailing <newline>) that describes the last error that occurred during dynamic linking processing. If no dynamic linking errors have occurred since the last invocation of *dlerror*(), *dlerror*() shall return NULL. Thus, invoking *dlerror*() a second time, immediately following a prior invocation, shall result in NULL being returned.

It is implementation-defined whether or not the *dlerror*() function is thread-safe. A thread-safe implementation shall return only errors that occur on the current thread.

RETURN VALUE

If successful, *dlerror*() shall return a null-terminated character string; otherwise, NULL shall be returned.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *dlerror*() in the same thread (if *dlerror*() is thread-safe) or in any thread (if *dlerror*() is not thread-safe). The returned pointer might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example prints out the last dynamic linking error:

```
...
#include <dlfcn.h>

char *errstr;

errstr = dlerror();
if (errstr != NULL)
    printf ("A dynamic linking error occurred: (%s)\n", errstr);
...
```

APPLICATION USAGE

Depending on the application environment with respect to asynchronous execution events, such as signals or other asynchronous computation sharing the address space, conforming applications should use a critical section to retrieve the error pointer and buffer.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlopen*( ), *dlsym*( )

The Base Definitions volume of POSIX.1-2017, **<dlfcn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dlopen — open a symbol table handle

SYNOPSIS

#include <dlfcn.h>

void *dlopen(const char **file*, int *mode*);

DESCRIPTION

The *dlopen*() function shall make the symbols (function identifiers and data object identifiers) in the executable object file specified by *file* available to the calling program.

The class of executable object files eligible for this operation and the manner of their construction are implementation-defined, though typically such files are shared libraries or programs.

Implementations may permit the construction of embedded dependencies in executable object files. In such cases, a *dlopen*() operation shall load those dependencies in addition to the executable object file specified by *file*. Implementations may also impose specific constraints on the construction of programs that can employ *dlopen*() and its related services.

A successful *dlopen*() shall return a symbol table handle which the caller may use on subsequent calls to *dlsym*() and *dlclose*().

The value of this symbol table handle should not be interpreted in any way by the caller.

The *file* argument is used to construct a pathname to the executable object file. If *file* contains a <slash> character, the *file* argument is used as the pathname for the file. Otherwise, *file* is used in an implementation-defined manner to yield a pathname.

If *file* is a null pointer, *dlopen*() shall return a global symbol table handle for the currently running process image. This symbol table handle shall provide access to the symbols from an ordered set of executable object files consisting of the original program image file, any executable object files loaded at program startup as specified by that process file (for example, shared libraries), and the set of executable object files loaded using *dlopen*() operations with the RTLD_GLOBAL flag. As the latter set of executable object files can change during execution, the set of symbols made available by this symbol table handle can also change dynamically.

Only a single copy of an executable object file shall be brought into the address space, even if *dlopen*() is invoked multiple times in reference to the executable object file, and even if different pathnames are used to reference the executable object file.

The *mode* parameter describes how *dlopen*() shall operate upon *file* with respect to the processing of relocations and the scope of visibility of the symbols provided within *file*. When an executable object file is brought into the address space of a process, it may contain references to symbols whose addresses are not known until the executable object file is loaded.

These references shall be relocated before the symbols can be accessed. The *mode* parameter governs when these relocations take place and may have the following values:

RTLD_LAZY
: Relocations shall be performed at an implementation-defined time, ranging from the time of the *dlopen*() call until the first reference to a given symbol occurs. Specifying RTLD_LAZY should improve performance on implementations supporting dynamic symbol binding since a process might not reference all of the symbols in an executable object file. And, for systems supporting dynamic symbol resolution for normal process execution, this behavior mimics the normal handling of process execution.

RTLD_NOW All necessary relocations shall be performed when the executable object file is first loaded. This may waste some processing if relocations are performed for symbols that are never referenced. This behavior may be useful for applications that need to know that all symbols referenced during execution will be available before *dlopen*() returns.

Any executable object file loaded by *dlopen*() that requires relocations against global symbols can reference the symbols in the original process image file, any executable object files loaded at program start-up, from the initial process image itself, from any other executable object file included in the same *dlopen*() invocation, and any executable object files that were loaded in any *dlopen*() invocation and which specified the RTLD_GLOBAL flag. To determine the scope of visibility for the symbols loaded with a *dlopen*() invocation, the *mode* parameter should be a bitwise-inclusive OR with one of the following values:

RTLD_GLOBAL
The executable object file's symbols shall be made available for relocation processing of any other executable object file. In addition, symbol lookup using *dlopen*(NULL,*mode*) and an associated *dlsym*() allows executable object files loaded with this mode to be searched.

RTLD_LOCAL
The executable object file's symbols shall not be made available for relocation processing of any other executable object file.

If neither RTLD_GLOBAL nor RTLD_LOCAL is specified, the default behavior is unspecified.

If an executable object file is specified in multiple *dlopen*() invocations, *mode* is interpreted at each invocation.

If RTLD_NOW has been specified, all relocations shall have been completed rendering further RTLD_NOW operations redundant and any further RTLD_LAZY operations irrelevant.

If RTLD_GLOBAL has been specified, the executable object file shall maintain the RTLD_GLOBAL status regardless of any previous or future specification of RTLD_LOCAL, as long as the executable object file remains in the address space (see *dlclose*( )).

Symbols introduced into the process image through calls to *dlopen*() may be used in relocation activities. Symbols so introduced may duplicate symbols already defined by the program or previous *dlopen*() operations. To resolve the ambiguities such a situation might present, the resolution of a symbol reference to symbol definition is based on a symbol resolution order. Two such resolution orders are defined: load order and dependency order. Load order establishes an ordering among symbol definitions, such that the first definition loaded (including definitions from the process image file and any dependent executable object files loaded with it) has priority over executable object files added later (by *dlopen*()). Load ordering is used in relocation processing. Dependency ordering uses a breadth-first order starting with a given executable object file, then all of its dependencies, then any dependents of those, iterating until all dependencies are satisfied. With the exception of the global symbol table handle obtained via a *dlopen*() operation with a null pointer as the *file* argument, dependency ordering is used by the *dlsym*() function. Load ordering is used in *dlsym*() operations upon the global symbol table handle.

When an executable object file is first made accessible via *dlopen*(), it and its dependent executable object files are added in dependency order. Once all the executable object files are added, relocations are performed using load order. Note that if an executable object file or its dependencies had been previously loaded, the load and dependency orders may yield different resolutions.

The symbols introduced by *dlopen*() operations and available through *dlsym*() are at a minimum those which are exported as identifiers of global scope by the executable object file. Typically, such identifiers shall be those that were specified in (for example) C source code as having **extern** linkage. The precise manner in which an implementation constructs the set of exported symbols for an executable object file is implementation-defined.

RETURN VALUE

Upon successful completion, *dlopen*() shall return a symbol table handle. If *file* cannot be found, cannot be opened for reading, is not of an appropriate executable object file format for processing by *dlopen*(), or if an error occurs during the process of loading *file* or relocating its symbolic references, *dlopen*() shall return a null pointer. More detailed diagnostic information shall be available through *dlerror*().

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Refer to *dlsym*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlerror*( ), *dlsym*( )

The Base Definitions volume of POSIX.1-2017, **<dlfcn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dlsym — get the address of a symbol from a symbol table handle

SYNOPSIS

#include <dlfcn.h>

void *dlsym(void *restrict *handle*, const char *restrict *name*);

DESCRIPTION

The *dlsym*() function shall obtain the address of a symbol (a function identifier or a data object identifier) defined in the symbol table identified by the *handle* argument. The *handle* argument is a symbol table handle returned from a call to *dlopen*() (and which has not since been released by a call to *dlclose*()), and *name* is the symbol's name as a character string. The return value from *dlsym*(), cast to a pointer to the type of the named symbol, can be used to call (in the case of a function) or access the contents of (in the case of a data object) the named symbol.

The *dlsym*() function shall search for the named symbol in the symbol table referenced by *handle*. If the symbol table was created with lazy loading (see RTLD_LAZY in *dlopen*()), load ordering shall be used in *dlsym*() operations to relocate executable object files needed to resolve the symbol. The symbol resolution algorithm used shall be dependency order as described in *dlopen*().

The RTLD_DEFAULT and RTLD_NEXT symbolic constants (which may be defined in *<dlfcn.h>*) are reserved for future use as special values that applications may be allowed to use for *handle*.

RETURN VALUE

Upon successful completion, if *name* names a function identifier, *dlsym*() shall return the address of the function converted from type pointer to function to type pointer to **void**; otherwise, *dlsym*() shall return the address of the data object associated with the data object identifier named by *name* converted from a pointer to the type of the data object to a pointer to **void**. If *handle* does not refer to a valid symbol table handle or if the symbol named by *name* cannot be found in the symbol table associated with *handle*, *dlsym*() shall return a null pointer.

More detailed diagnostic information shall be available through *dlerror*().

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example shows how *dlopen*() and *dlsym*() can be used to access either a function or a data object. For simplicity, error checking has been omitted.

```
void *handle;
int (*fptr)(int), *iptr, result;
/* open the needed symbol table */
handle = dlopen("/usr/home/me/libfoo.so", RTLD_LOCAL | RTLD_LAZY);
/* find the address of the function my_function */
fptr = (int (*)(int))dlsym(handle, "my_function");
/* find the address of the data object my_object */
iptr = (int *)dlsym(handle, "my_OBJ");
/* invoke my_function, passing the value of my_OBJ as the parameter */
result = (*fptr)(*iptr);
```

APPLICATION USAGE

The following special purpose values for *handle* are reserved for future use and have the indicated meanings:

RTLD_DEFAULT
: The identifier lookup happens in the normal global scope; that is, a search for an identifier using *handle* would find the same definition as a direct use of this identifier in the program code.

RTLD_NEXT
: Specifies the next executable object file after this one that defines *name*. This one refers to the executable object file containing the invocation of *dlsym*(). The next executable object file is the one found upon the application of a load order symbol resolution algorithm (see *dlopen*()). The next symbol is either one of global scope (because it was introduced as part of the original process image or because it was added with a *dlopen*() operation including the RTLD_GLOBAL flag), or is in an executable object file that was included in the same *dlopen*() operation that loaded this one.

The RTLD_NEXT flag is useful to navigate an intentionally created hierarchy of multiply-defined symbols created through interposition. For example, if a program wished to create an implementation of *malloc*() that embedded some statistics gathering about memory allocations, such an implementation could use the real *malloc*() definition to perform the memory allocation — and itself only embed the necessary logic to implement the statistics gathering function.

Note that conversion from a **void *** pointer to a function pointer as in:

```
fptr = (int (*)(int))dlsym(handle, "my_function");
```

is not defined by the ISO C standard. This standard requires this conversion to work correctly on conforming implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

dlclose( ), *dlerror*( ), *dlopen*( )

The Base Definitions volume of POSIX.1-2017, **<dlfcn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>

int dprintf(int fildes, const char *restrict format, ...);
```

DESCRIPTION

Refer to *fprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

drand48, erand48, jrand48, lcong48, lrand48, mrand48, nrand48, seed48, srand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

#include <stdlib.h>

double drand48(void);
double erand48(unsigned short *xsubi*[3]);
long jrand48(unsigned short *xsubi*[3]);
void lcong48(unsigned short *param*[7]);
long lrand48(void);
long mrand48(void);
long nrand48(unsigned short *xsubi*[3]);
unsigned short *seed48(unsigned short *seed16v*[3]);
void srand48(long *seedval*);

DESCRIPTION

This family of functions shall generate pseudo-random numbers using a linear congruential algorithm and 48-bit integer arithmetic.

The *drand48*() and *erand48*() functions shall return non-negative, double-precision, floating-point values, uniformly distributed over the interval [0.0,1.0).

The *lrand48*() and *nrand48*() functions shall return non-negative, long integers, uniformly distributed over the interval $[0,2^{31})$.

The *mrand48*() and *jrand48*() functions shall return signed long integers uniformly distributed over the interval $[-2^{31},2^{31})$.

The *srand48*(), *seed48*(), and *lcong48*() functions are initialization entry points, one of which should be invoked before either *drand48*(), *lrand48*(), or *mrand48*() is called. (Although it is not recommended practice, constant default initializer values shall be supplied automatically if *drand48*(), *lrand48*(), or *mrand48*() is called without a prior call to an initialization entry point.) The *erand48*(), *nrand48*(), and *jrand48*() functions do not require an initialization entry point to be called first.

All the routines work by generating a sequence of 48-bit integer values, $X_ i" " ,$ according to the linear congruential formula:

> $X sub{n+1} " " = " " (aX_ n" "^+^c) sub{roman mod " " m} " " " " " " " " " " " " " " " " n>= " " 0$

The parameter $m^=^2"^" 48$; hence 48-bit integer arithmetic is performed. Unless *lcong48*() is invoked, the multiplier value a and the addend value c are given by:

> $a " " mark = " " roman "5DEECE66D"^sub 16 " " = " " roman 273673163155^sub 8$
>
> $c " " lineup = " " roman B^sub 16 " " = " " roman 13^sub 8$

The value returned by any of the *drand48*(), *erand48*(), *jrand48*(), *lrand48*(), *mrand48*(), or *nrand48*() functions is computed by first generating the next 48-bit $X_ i$ in the sequence. Then the appropriate number of bits, according to the type of data item to be returned, are copied from the high-order (leftmost) bits of $X_ i$ and transformed into the returned value.

The *drand48*(), *lrand48*(), and *mrand48*() functions store the last 48-bit $X_ i$ generated in an internal buffer; that is why the application shall ensure that these are initialized prior to being invoked. The *erand48*(), *nrand48*(), and *jrand48*() functions require the calling program to provide storage for the successive $X_ i$

values in the array specified as an argument when the functions are invoked. That is why these routines do not have to be initialized; the calling program merely has to place the desired initial value of $X_ i$ into the array and pass it as an argument. By using different arguments, *erand48*(), *nrand48*(), and *jrand48*() allow separate modules of a large program to generate several *independent* streams of pseudo-random numbers; that is, the sequence of numbers in each stream shall *not* depend upon how many times the routines are called to generate numbers for the other streams.

The initializer function *srand48*() sets the high-order 32 bits of $X_ i$ to the low-order 32 bits contained in its argument. The low-order 16 bits of $X_ i$ are set to the arbitrary value $roman 330E_ 16" " .$

The initializer function *seed48*() sets the value of $X_ i$ to the 48-bit value specified in the argument array. The low-order 16 bits of $X_ i$ are set to the low-order 16 bits of *seed16v*[*0*]. The mid-order 16 bits of $X_ i$ are set to the low-order 16 bits of *seed16v*[*1*]. The high-order 16 bits of $X_ i$ are set to the low-order 16 bits of *seed16v*[*2*]. In addition, the previous value of $X_ i$ is copied into a 48-bit internal buffer, used only by *seed48*(), and a pointer to this buffer is the value returned by *seed48*(). This returned pointer, which can just be ignored if not needed, is useful if a program is to be restarted from a given point at some future time—use the pointer to get at and store the last $X_ i$ value, and then use this value to reinitialize via *seed48*() when the program is restarted.

The initializer function *lcong48*() allows the user to specify the initial $X_ i" " ,$ the multiplier value $a,$ and the addend value $c.$ Argument array elements *param*[*0-2*] specify $X_ i" " ,$ *param*[*3-5*] specify the multiplier $a,$ and *param*[*6*] specifies the 16-bit addend $c.$ After *lcong48*() is called, a subsequent call to either *srand48*() or *seed48*() shall restore the standard multiplier and addend values, *a* and *c,* specified above.

The *drand48*(), *lrand48*(), and *mrand48*() functions need not be thread-safe.

RETURN VALUE

As described in the DESCRIPTION above.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

initstate( ), *rand*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dup, dup2 — duplicate an open file descriptor

SYNOPSIS

```
#include <unistd.h>

int dup(int fildes);
int dup2(int fildes, int fildes2);
```

DESCRIPTION

The *dup*() function provides an alternative interface to the service provided by *fcntl*() using the F_DUPFD command. The call *dup*(*fildes*) shall be equivalent to:

```
fcntl(fildes, F_DUPFD, 0);
```

The *dup2*() function shall cause the file descriptor *fildes2* to refer to the same open file description as the file descriptor *fildes* and to share any locks, and shall return *fildes2*. If *fildes2* is already a valid open file descriptor, it shall be closed first, unless *fildes* is equal to *fildes2* in which case *dup2*() shall return *fildes2* without closing it. If the close operation fails to close *fildes2*, *dup2*() shall return −1 without changing the open file description to which *fildes2* refers. If *fildes* is not a valid file descriptor, *dup2*() shall return −1 and shall not close *fildes2*. If *fildes2* is less than 0 or greater than or equal to {OPEN_MAX}, *dup2*() shall return −1 with *errno* set to **[EBADF]**.

Upon successful completion, if *fildes* is not equal to *fildes2*, the FD_CLOEXEC flag associated with *fildes2* shall be cleared. If *fildes* is equal to *fildes2*, the FD_CLOEXEC flag associated with *fildes2* shall not be changed.

If *fildes* refers to a typed memory object, the result of the *dup2*() function is unspecified.

RETURN VALUE

Upon successful completion a non-negative integer, namely the file descriptor, shall be returned; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *dup*() function shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor.

EMFILE
All file descriptors available to the process are currently open.

The *dup2*() function shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor or the argument *fildes2* is negative or greater than or equal to {OPEN_MAX}.

EINTR
The *dup2*() function was interrupted by a signal.

The *dup2*() function may fail if:

EIO An I/O error occurred while attempting to close *fildes2*.

The following sections are informative.

EXAMPLES

Redirecting Standard Output to a File S

The following example closes standard output for the current processes, re-assigns standard output to go to the file referenced by *pfd*, and closes the original file descriptor to clean up.

```
#include <unistd.h>
...
int pfd;
...
close(1);
dup(pfd);
close(pfd);
...
```

Redirecting Error Messages

The following example redirects messages from *stderr* to *stdout*.

```
#include <unistd.h>
...
dup2(1, 2);
...
```

APPLICATION USAGE

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *dup2*() with an arbitrary integer for *fildes2* risks non-conforming behavior, and *dup2*() can only portably be used to overwrite file descriptor values that the application has obtained through explicit actions, or for the three file descriptors corresponding to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process, an application should consider opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

RATIONALE

The *dup*() function is redundant. Its services are also provided by the *fcntl*() function. It has been included in this volume of POSIX.1-2017 primarily for historical reasons, since many existing applications use it. On the other hand, the *dup2*() function provides unique services, as no other interface is able to atomically replace an existing file descriptor.

The *dup2*() function is not marked obsolescent because it presents a type-safe version of functionality provided in a type-unsafe version by *fcntl*(). It is used in the POSIX Ada binding.

The *dup2*() function is not intended for use in critical regions as a synchronization mechanism.

In the description of **[EBADF]**, the case of *fildes* being out of range is covered by the given case of *fildes* not being valid. The descriptions for *fildes* and *fildes2* are different because the only kind of invalidity that is relevant for *fildes2* is whether it is out of range; that is, it does not matter whether *fildes2* refers to an open file when the *dup2*() call is made.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *fcntl*( ), *open*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

duplocale — duplicate a locale object

SYNOPSIS

#include <locale.h>

locale_t duplocale(locale_t *locobj*);

DESCRIPTION

The *duplocale*() function shall create a duplicate copy of the locale object referenced by the *locobj* argument.

If the *locobj* argument is LC_GLOBAL_LOCALE, *duplocale*() shall create a new locale object containing a copy of the global locale determined by the *setlocale*() function.

The behavior is undefined if the *locobj* argument is not a valid locale object handle.

RETURN VALUE

Upon successful completion, the *duplocale*() function shall return a handle for a new locale object. Otherwise, *duplocale*() shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *duplocale*() function shall fail if:

ENOMEM
There is not enough memory available to create the locale object or load the locale data.

The following sections are informative.

EXAMPLES

Constructing an Altered Version of an Existing Locale Object

The following example shows a code fragment to create a slightly altered version of an existing locale object. The function takes a locale object and a locale name and it replaces the *LC_TIME* category data in the locale object with that from the named locale.

```
#include <locale.h>
...

locale_t
with_changed_lc_time (locale_t obj, const char *name)
{

  locale_t retval = duplocale (obj);
  if (retval != (locale_t) 0)
  {
    locale_t changed = newlocale (LC_TIME_MASK, name, retval);
    if (changed == (locale_t) 0)
      /* An error occurred. Free all allocated resources. */
      freelocale (retval);
    retval = changed;
  }
  return retval;
}
```

APPLICATION USAGE

The use of the *duplocale*() function is recommended for situations where a locale object is being used in multiple places, and it is possible that the lifetime of the locale object might end before all uses are finished.

Another reason to duplicate a locale object is if a slightly modified form is needed. This can be achieved by a call to *newlocale*() following the *duplocale*() call.

As with the *newlocale*() function, handles for locale objects created by the *duplocale*() function should be released by a corresponding call to *freelocale*().

The *duplocale*() function can also be used in conjunction with *uselocale*((**locale_t**)0). This returns the locale in effect for the calling thread, but can have the value LC_GLOBAL_LOCALE. Passing LC_GLOBAL_LOCALE to functions such as *isalnum_l*() results in undefined behavior, but applications can convert it into a usable locale object by using *duplocale*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

freelocale( ), *newlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

encrypt — encoding function (**CRYPT**)

SYNOPSIS

```
#include <unistd.h>

void encrypt(char block[64], int edflag);
```

DESCRIPTION

The *encrypt*() function shall provide access to an implementation-defined encoding algorithm. The key generated by *setkey*() is used to encrypt the string *block* with *encrypt*().

The *block* argument to *encrypt*() shall be an array of length 64 bytes containing only the bytes with values of 0 and 1. The array is modified in place to a similar array using the key set by *setkey*(). If *edflag* is 0, the argument is encoded. If *edflag* is 1, the argument may be decoded (see the APPLICATION USAGE section); if the argument is not decoded, *errno* shall be set to **[ENOSYS]**.

The *encrypt*() function shall not change the setting of *errno* if successful. An application wishing to check for error situations should set *errno* to 0 before calling *encrypt*(). If *errno* is non-zero on return, an error has occurred.

The *encrypt*() function need not be thread-safe.

RETURN VALUE

The *encrypt*() function shall not return a value.

ERRORS

The *encrypt*() function shall fail if:

ENOSYS
The functionality is not supported on this implementation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Historical implementations of the *encrypt*() function used a rather primitive encoding algorithm.

In some environments, decoding might not be implemented. This is related to some Government restrictions on encryption and decryption routines. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of *encrypt*() does encoding but not decoding.

RATIONALE

None.

FUTURE DIRECTIONS

A future version of the standard may mark this interface as obsolete or remove it altogether.

SEE ALSO

crypt( ), *setkey*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endgrent, getgrent, setgrent — group database entry functions

SYNOPSIS

```
#include <grp.h>

void endgrent(void);
struct group *getgrent(void);
void setgrent(void);
```

DESCRIPTION

The *getgrent*() function shall return a pointer to a structure containing the broken-out fields of an entry in the group database. If the group database is not already open, *getgrent*() shall open it and return a pointer to a **group** structure containing the first entry in the database. Thereafter, it shall return a pointer to a **group** structure containing the next **group** structure in the group database, so successive calls may be used to search the entire database.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the group database. In particular, the system may deny the existence of some or all of the group database entries associated with groups other than those groups associated with the caller and may omit users other than the caller from the list of members of groups in database entries that are returned.

The *setgrent*() function shall rewind the group database so that the next *getgrent*() call returns the first entry, allowing repeated searches.

The *endgrent*() function shall close the group database.

The *setgrent*() and *endgrent*() functions shall not change the setting of *errno* if successful.

On error, the *setgrent*() and *endgrent*() functions shall set *errno* to indicate the error.

Since no value is returned by the *setgrent*() and *endgrent*() functions, an application wishing to check for error situations should set *errno* to 0, then call the function, then check *errno*.

These functions need not be thread-safe.

RETURN VALUE

On successful completion, *getgrent*() shall return a pointer to a **group** structure. On end-of-file, *getgrent*() shall return a null pointer and shall not change the setting of *errno*. On error, *getgrent*() shall return a null pointer and *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getgrgid*(), *getgrnam*(), or *getgrent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

These functions may fail if:

EINTR
A signal was caught during the operation.

EIO An I/O error has occurred.

In addition, the *getgrent*() and *setgrent*() functions may fail if:

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the group database, whether the database is a single file, or where in the file system name space the database resides. Applications should use *getgrnam*() and *getgrgid*() whenever possible because it avoids these dependencies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endpwent( ), *getgrgid*( ), *getgrnam*( ), *getlogin*( )

The Base Definitions volume of POSIX.1-2017, **<grp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endhostent, gethostent, sethostent — network host database functions

SYNOPSIS

```
#include <netdb.h>

void endhostent(void);
struct hostent *gethostent(void);
void sethostent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about hosts. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

Note: In many cases this database is implemented by the Domain Name System, as documented in RFC 1034, RFC 1035, and RFC 1886.

The *sethostent*() function shall open a connection to the database and set the next entry for retrieval to the first entry in the database. If the *stayopen* argument is non-zero, the connection shall not be closed by a call to *gethostent*(), and the implementation may maintain an open file descriptor.

The *gethostent*() function shall read the next entry in the database, opening and closing a connection to the database as necessary.

Entries shall be returned in **hostent** structures.

The *endhostent*() function shall close the connection to the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, the *gethostent*() function shall return a pointer to a **hostent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *gethostent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined for *endhostent*(), *gethostent*(), and *sethostent*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endservent( )

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endnetent, getnetbyaddr, getnetbyname, getnetent, setnetent — network database functions

SYNOPSIS

```
#include <netdb.h>

void endnetent(void);
struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
void setnetent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about networks. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setnetent*() function shall open and rewind the database. If the *stayopen* argument is non-zero, the connection to the *net* database shall not be closed after each call to *getnetent*() (either directly, or indirectly through one of the other *getnet**( ) functions), and the implementation may maintain an open file descriptor to the database.

The *getnetent*() function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getnetbyaddr*() function shall search the database from the beginning, and find the first entry for which the address family specified by *type* matches the *n_addrtype* member and the network number *net* matches the *n_net* member, opening and closing a connection to the database as necessary. The *net* argument shall be the network number in host byte order.

The *getnetbyname*() function shall search the database from the beginning and find the first entry for which the network name specified by *name* matches the *n_name* member, opening and closing a connection to the database as necessary.

The *getnetbyaddr*(), *getnetbyname*(), and *getnetent*() functions shall each return a pointer to a **netent** structure, the members of which shall contain the fields of an entry in the network database.

The *endnetent*() function shall close the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getnetbyaddr*(), *getnetbyname*(), and *getnetent*() shall return a pointer to a **netent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer shall be returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getnetbyaddr*(), *getnetbyname*(), or *getnetent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endprotoent, getprotobyname, getprotobynumber, getprotoent, setprotoent — network protocol database functions

SYNOPSIS

#include <netdb.h>

void endprotoent(void);
struct protoent *getprotobyname(const char **name*);
struct protoent *getprotobynumber(int *proto*);
struct protoent *getprotoent(void);
void setprotoent(int *stayopen*);

DESCRIPTION

These functions shall retrieve information about protocols. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setprotoent*() function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the connection to the network protocol database shall not be closed after each call to *getprotoent*() (either directly, or indirectly through one of the other *getproto**( ) functions), and the implementation may maintain an open file descriptor for the database.

The *getprotobyname*() function shall search the database from the beginning and find the first entry for which the protocol name specified by *name* matches the *p_name* member, opening and closing a connection to the database as necessary.

The *getprotobynumber*() function shall search the database from the beginning and find the first entry for which the protocol number specified by *proto* matches the *p_proto* member, opening and closing a connection to the database as necessary.

The *getprotoent*() function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getprotobyname*(), *getprotobynumber*(), and *getprotoent*() functions shall each return a pointer to a **protoent** structure, the members of which shall contain the fields of an entry in the network protocol database.

The *endprotoent*() function shall close the connection to the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getprotobyname*(), *getprotobynumber*(), and *getprotoent*() return a pointer to a **protoent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getprotobyname*(), *getprotobynumber*(), or *getprotoent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endpwent, getpwent, setpwent — user database functions

SYNOPSIS

```
#include <pwd.h>

void endpwent(void);
struct passwd *getpwent(void);
void setpwent(void);
```

DESCRIPTION

These functions shall retrieve information about users.

The *getpwent*() function shall return a pointer to a structure containing the broken-out fields of an entry in the user database. Each entry in the user database contains a **passwd** structure. If the user database is not already open, *getpwent*() shall open it and return a pointer to a **passwd** structure containing the first entry in the database. Thereafter, it shall return a pointer to a **passwd** structure containing the next entry in the user database. Successive calls can be used to search the entire user database.

If an end-of-file or an error is encountered on reading, *getpwent*() shall return a null pointer.

An implementation that provides extended security controls may impose further implementation-defined restrictions on accessing the user database. In particular, the system may deny the existence of some or all of the user database entries associated with users other than the caller.

The *setpwent*() function shall rewind the user database so that the next *getpwent*() call returns the first entry, allowing repeated searches.

The *endpwent*() function shall close the user database.

The *setpwent*() and *endpwent*() functions shall not change the setting of *errno* if successful.

On error, the *setpwent*() and *endpwent*() functions shall set *errno* to indicate the error.

Since no value is returned by the *setpwent*() and *endpwent*() functions, an application wishing to check for error situations should set *errno* to 0, then call the function, then check *errno*.

These functions need not be thread-safe.

RETURN VALUE

On successful completion, *getpwent*() shall return a pointer to a **passwd** structure. On end-of-file, *getpwent*() shall return a null pointer and shall not change the setting of *errno*. On error, *getpwent*() shall return a null pointer and *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getpwuid*(), *getpwnam*(), or *getpwent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

These functions may fail if:

EINTR
A signal was caught during the operation.

EIO An I/O error has occurred.

In addition, *getpwent*() and *setpwent*() may fail if:

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The following sections are informative.

EXAMPLES

Searching the User Database

The following example uses the *getpwent*() function to get successive entries in the user database, returning a pointer to a **passwd** structure that contains information about each user. The call to *endpwent*() closes the user database and cleans up.

```
#include <pwd.h>
#include <stdio.h>

void printname(uid_t uid)
{
   struct passwd *pwd;

   setpwent();
   while((pwd = getpwent()) != NULL) {
      if (pwd->pw_uid == uid) {
         printf("name=%s\n",pwd->pw_name);
         break;
      }
   }
   endpwent();
}
```

APPLICATION USAGE

These functions are provided due to their historical usage. Applications should avoid dependencies on fields in the password database, whether the database is a single file, or where in the file system name space the database resides. Applications should use *getpwuid*() whenever possible because it avoids these dependencies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getlogin*( ), *getpwnam*( ), *getpwuid*( )

The Base Definitions volume of POSIX.1-2017, **<pwd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endservent, getservbyname, getservbyport, getservent, setservent — network services database functions

SYNOPSIS

```
#include <netdb.h>

void endservent(void);
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
void setservent(int stayopen);
```

DESCRIPTION

These functions shall retrieve information about network services. This information is considered to be stored in a database that can be accessed sequentially or randomly. The implementation of this database is unspecified.

The *setservent*() function shall open a connection to the database, and set the next entry to the first entry. If the *stayopen* argument is non-zero, the *net* database shall not be closed after each call to the *getservent*() function (either directly, or indirectly through one of the other *getserv**( ) functions), and the implementation may maintain an open file descriptor for the database.

The *getservent*() function shall read the next entry of the database, opening and closing a connection to the database as necessary.

The *getservbyname*() function shall search the database from the beginning and find the first entry for which the service name specified by *name* matches the *s_name* member and the protocol name specified by *proto* matches the *s_proto* member, opening and closing a connection to the database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched.

The *getservbyport*() function shall search the database from the beginning and find the first entry for which the port specified by *port* matches the *s_port* member and the protocol name specified by *proto* matches the *s_proto* member, opening and closing a connection to the database as necessary. If *proto* is a null pointer, any value of the *s_proto* member shall be matched. The *port* argument shall be a value obtained by converting a **uint16_t** in network byte order to **int**.

The *getservbyname*(), *getservbyport*(), and *getservent*() functions shall each return a pointer to a **servent** structure, the members of which shall contain the fields of an entry in the network services database.

The *endservent*() function shall close the database, releasing any open file descriptor.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getservbyname*(), *getservbyport*(), and *getservent*() return a pointer to a **servent** structure if the requested entry was found, and a null pointer if the end of the database was reached or the requested entry was not found. Otherwise, a null pointer is returned.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getservbyname*(), *getservbyport*(), or *getservent*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *port* argument of *getservbyport*() need not be compatible with the port values of all address families.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endhostent( ), *endprotoent*( ), *htonl*( ), *inet_addr*( )

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

endutxent, getutxent, getutxid, getutxline, pututxline, setutxent — user accounting database functions

SYNOPSIS

```
#include <utmpx.h>

void endutxent(void);
struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
struct utmpx *pututxline(const struct utmpx *utmpx);
void setutxent(void);
```

DESCRIPTION

These functions shall provide access to the user accounting database.

The *getutxent*() function shall read the next entry from the user accounting database. If the database is not already open, it shall open it. If it reaches the end of the database, it shall fail.

The *getutxid*() function shall search forward from the current point in the database. If the *ut_type* value of the **utmpx** structure pointed to by *id* is BOOT_TIME, OLD_TIME, or NEW_TIME, then it shall stop when it finds an entry with a matching *ut_type* value. If the *ut_type* value is INIT_PROCESS, LOGIN_PROCESS, USER_PROCESS, or DEAD_PROCESS, then it shall stop when it finds an entry whose type is one of these four and whose *ut_id* member matches the *ut_id* member of the **utmpx** structure pointed to by *id*. If the end of the database is reached without a match, *getutxid*() shall fail.

The *getutxline*() function shall search forward from the current point in the database until it finds an entry of the type LOGIN_PROCESS or USER_PROCESS which also has a *ut_line* value matching that in the **utmpx** structure pointed to by *line*. If the end of the database is reached without a match, *getutxline*() shall fail.

The *getutxid*() or *getutxline*() function may cache data. For this reason, to use *getutxline*() to search for multiple occurrences, the application shall zero out the static data after each success, or *getutxline*() may return a pointer to the same **utmpx** structure.

There is one exception to the rule about clearing the structure before further reads are done. The implicit read done by *pututxline*() (if it finds that it is not already at the correct place in the user accounting database) shall not modify the static structure returned by *getutxent*(), *getutxid*(), or *getutxline*(), if the application has modified this structure and passed the pointer back to *pututxline*().

For all entries that match a request, the *ut_type* member indicates the type of the entry. Other members of the entry shall contain meaningful data based on the value of the *ut_type* member as follows:

box center tab(!); cB | cB l | l. ut_type Member!Other Members with Meaningful Data _ EMPTY!No others BOOT_TIME!*ut_tv* OLD_TIME!*ut_tv* NEW_TIME!*ut_tv* USER_PROCESS!*ut_id*, *ut_user* (login name of the user), *ut_line*, *ut_pid*, *ut_tv* INIT_PROCESS!*ut_id*, *ut_pid*, *ut_tv* LOGIN_PROCESS!T{ *ut_id*, *ut_user* (implementation-defined name of the login process), *ut_line*, *ut_pid*, *ut_tv* T} DEAD_PROCESS!*ut_id*, *ut_pid*, *ut_tv*

An implementation that provides extended security controls may impose implementation-defined restrictions on accessing the user accounting database. In particular, the system may deny the existence of some or all of the user accounting database entries associated with users other than the caller.

If the process has appropriate privileges, the *pututxline*() function shall write out the structure into the user accounting database. It shall search for a record as if by *getutxid*() that satisfies the request. If this search succeeds, then the entry shall be replaced. Otherwise, a new entry shall be made at the end of the user accounting database.

The *endutxent*() function shall close the user accounting database.

The *setutxent*() function shall reset the input to the beginning of the database. This should be done before each search for a new entry if it is desired that the entire database be examined.

These functions need not be thread-safe.

RETURN VALUE

Upon successful completion, *getutxent*(), *getutxid*(), and *getutxline*() shall return a pointer to a **utmpx** structure containing a copy of the requested entry in the user accounting database. Otherwise, a null pointer shall be returned.

The return value may point to a static area which is overwritten by a subsequent call to *getutxid*() or *getutxline*().

Upon successful completion, *pututxline*() shall return a pointer to a **utmpx** structure containing a copy of the entry added to the user accounting database. Otherwise, a null pointer shall be returned.

The *endutxent*() and *setutxent*() functions shall not return a value.

ERRORS

No errors are defined for the *endutxent*(), *getutxent*(), *getutxid*(), *getutxline*(), and *setutxent*() functions.

The *pututxline*() function may fail if:

EPERM
The process does not have appropriate privileges.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The sizes of the arrays in the structure can be found using the *sizeof* operator.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<utmpx.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

environ — array of character pointers to the environment strings

SYNOPSIS

extern char **environ;

DESCRIPTION

Refer to *exec* and the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*.

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

erand48 — generate uniformly distributed pseudo-random numbers

SYNOPSIS

```
#include <stdlib.h>

double erand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

erf, erff, erfl — error functions

SYNOPSIS

```
#include <math.h>

double erf(double x);
float erff(float x);
long double erfl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the error function of their argument *x*, defined as:

${2 over sqrt pi} int from 0 to x e"^" " "{- t"^" 2" "} dt$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the error function.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is ±Inf, ±1 shall be returned.

If the correct value would cause underflow, a range error may occur, and *erf*(), *erff*(), and *erfl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If the IEC 60559 Floating-Point option is supported, 2 * *x*/*sqrt*(*π*) should be returned.

ERRORS

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Computing the Probability for a Normal Variate

This example shows how to use *erf*() to compute the probability that a normal variate assumes a value in the range [*x*1,*x*2] with *x*1≤*x*2.

This example uses the constant M_SQRT1_2 which is part of the XSI option.

```
#include <math.h>

double
```

```
Phi(const double x1, const double x2)
{
   return ( erf(x2*M_SQRT1_2) - erf(x1*M_SQRT1_2) ) / 2;
}
```

APPLICATION USAGE

Underflow occurs when $|x|$ < DBL_MIN * (*sqrt*(π)/2).

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erfc( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

erfc, erfcf, erfcl — complementary error functions

SYNOPSIS

```
#include <math.h>

double erfc(double x);
float erfcf(float x);
long double erfcl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the complementary error function 1.0 − *erf*(*x*).

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of the complementary error function.

If the correct value would cause underflow, and is not representable, a range error may occur, and *erfc*(), *erfcf*(), and *erfcl*() shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, +1 shall be returned.

If *x* is −Inf, +2 shall be returned.

If *x* is +Inf, +0 shall be returned.

If the correct value would cause underflow and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *erfc*() function is provided because of the extreme loss of relative accuracy if *erf*(*x*) is called for large *x* and the result subtracted from 1.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

erf( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

erff, erfl — error functions

SYNOPSIS

```
#include <math.h>

float erff(float x);
long double erfl(long double x);
```

DESCRIPTION

Refer to *erf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

errno — error return value

SYNOPSIS

```
#include <errno.h>
```

DESCRIPTION

The lvalue *errno* is used by many functions to return error values.

Many functions provide an error number in *errno*, which has type **int** and is defined in *<errno.h>*. The value of *errno* shall be defined only after a call to a function for which it is explicitly stated to be set and until it is changed by the next function call or if the application assigns it a value. The value of *errno* should only be examined when it is indicated to be valid by a function's return value. Applications shall obtain the definition of *errno* by the inclusion of *<errno.h>*. No function in this volume of POSIX.1-2017 shall set *errno* to 0. The setting of *errno* after a successful call to a function is unspecified unless the description of that function specifies that *errno* shall not be modified.

It is unspecified whether *errno* is a macro or an identifier declared with external linkage. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *errno*, the behavior is undefined.

The symbolic values stored in *errno* are documented in the ERRORS sections on all relevant pages.

RETURN VALUE

None.

ERRORS

None.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Previously both POSIX and X/Open documents were more restrictive than the ISO C standard in that they required *errno* to be defined as an external variable, whereas the ISO C standard required only that *errno* be defined as a modifiable lvalue with type **int**.

An application that needs to examine the value of *errno* to determine the error should set it to 0 before a function call, then inspect it before a subsequent function call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3, *Error Numbers*

The Base Definitions volume of POSIX.1-2017, **<errno.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

environ, execl, execle, execlp, execv, execve, execvp, fexecve — execute a file

SYNOPSIS

```
#include <unistd.h>

extern char **environ;
int execl(const char *path, const char *arg0, ... /*, (char *)0 */);
int execle(const char *path, const char *arg0, ... /*,
    (char *)0, char *const envp[]*/);
int execlp(const char *file, const char *arg0, ... /*, (char *)0 */);
int execv(const char *path, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
int execvp(const char *file, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

DESCRIPTION

The *exec* family of functions shall replace the current process image with a new process image. The new image shall be constructed from a regular, executable file called the *new process image file*. There shall be no return from a successful *exec*, because the calling process image is overlaid by the new process image.

The *fexecve*() function shall be equivalent to the *execve*() function except that the file to be executed is determined by the file descriptor *fd* instead of a pathname. The file offset of *fd* is ignored.

When a C-language program is executed as a result of a call to one of the *exec* family of functions, it shall be entered as a C-language function call as follows:

```
int main (int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable, which must be declared by the user if it is to be used directly:

```
extern char **environ;
```

is initialized as a pointer to an array of character pointers to the environment strings. The *argv* and *environ* arrays are each terminated by a null pointer. The null pointer terminating the *argv* array is not counted in *argc*.

Applications can change the entire environment in a single operation by assigning the *environ* variable to point to an array of character pointers to the new environment strings. After assigning a new value to *environ*, applications should not rely on the new environment strings remaining part of the environment, as a call to *getenv*(), *putenv*(), *setenv*(), *unsetenv*(), or any function that is dependent on an environment variable may, on noticing that *environ* has changed, copy the environment strings to a new array and assign *environ* to point to it.

Any application that directly modifies the pointers to which the *environ* variable points has undefined behavior.

Conforming multi-threaded applications shall not use the *environ* variable to access or modify any environment variable while any other thread is concurrently modifying any environment variable. A call to any function dependent on any environment variable shall be considered a use of the *environ* variable to access that environment variable.

The arguments specified by a program with one of the *exec* functions shall be passed on to the new process image in the corresponding *main*() arguments.

The argument *path* points to a pathname that identifies the new process image file.

The argument *file* is used to construct a pathname that identifies the new process image file. If the *file* argument contains a <slash> character, the *file* argument shall be used as the pathname for this file. Otherwise, the path prefix for this file is obtained by a search of the directories passed as the environment variable *PATH* (see the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*). If this environment variable is not present, the results of the search are implementation-defined.

There are two distinct ways in which the contents of the process image file may cause the execution to fail, distinguished by the setting of *errno* to either **[ENOEXEC]** or **[EINVAL]** (see the ERRORS section). In the cases where the other members of the *exec* family of functions would fail and set *errno* to **[ENOEXEC]**, the *execlp*() and *execvp*() functions shall execute a command interpreter and the environment of the executed command shall be as if the process invoked the *sh* utility using *execl*() as follows:

```
    execl(<shell path>, arg0, file, arg1, ..., (char *)0);
```

where <*shell path*> is an unspecified pathname for the *sh* utility, *file* is the process image file, and for *execvp*(), where *arg*0, *arg*1, and so on correspond to the values passed to *execvp*() in *argv*[0], *argv*[1], and so on.

The arguments represented by *arg0*,... are pointers to null-terminated character strings. These strings shall constitute the argument list available to the new process image. The list is terminated by a null pointer. The argument *arg0* should point to a filename string that is associated with the process being started by one of the *exec* functions.

The argument *argv* is an array of character pointers to null-terminated strings. The application shall ensure that the last member of this array is a null pointer. These strings shall constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename string that is associated with the process being started by one of the *exec* functions.

The argument *envp* is an array of character pointers to null-terminated strings. These strings shall constitute the environment for the new process image. The *envp* array is terminated by a null pointer.

For those forms not containing an *envp* pointer (*execl*(), *execv*(), *execlp*(), and *execvp*()), the environment for the new process image shall be taken from the external variable *environ* in the calling process.

The number of bytes available for the new process' combined argument and environment lists is {ARG_MAX}. It is implementation-defined whether null terminators, pointers, and/or any alignment bytes are included in this total.

File descriptors open in the calling process image shall remain open in the new process image, except for those whose close-on-*exec* flag FD_CLOEXEC is set. For those file descriptors that remain open, all attributes of the open file description remain unchanged. For any file descriptor that is closed for this reason, file locks are removed as a result of the close as described in *close*(). Locks that are not removed by closing of file descriptors remain unchanged.

If file descriptor 0, 1, or 2 would otherwise be closed after a successful call to one of the *exec* family of functions, implementations may open an unspecified file for the file descriptor in the new process image. If a standard utility or a conforming application is executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the environment in which the utility or application is executed shall be deemed non-conforming, and consequently the utility or application might not behave as described in this standard.

Directory streams open in the calling process image shall be closed in the new process image.

The state of the floating-point environment in the initial thread of the new process image shall be set to the default.

The state of conversion descriptors and message catalog descriptors in the new process image is undefined.

For the new process image, the equivalent of:

```
setlocale(LC_ALL, "C")
```

shall be executed at start-up.

Signals set to the default action (SIG_DFL) in the calling process image shall be set to the default action in the new process image. Except for SIGCHLD, signals set to be ignored (SIG_IGN) by the calling process image shall be set to be ignored by the new process image. Signals set to be caught by the calling process image shall be set to the default action in the new process image (see *<signal.h>*).

If the SIGCHLD signal is set to be ignored by the calling process image, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the new process image.

After a successful call to any of the *exec* functions, alternate signal stacks are not preserved and the SA_ONSTACK flag shall be cleared for all signals.

After a successful call to any of the *exec* functions, any functions previously registered by the *atexit*() or *pthread_atfork*() functions are no longer registered.

If the ST_NOSUID bit is set for the file system containing the new process image file, then the effective user ID, effective group ID, saved set-user-ID, and saved set-group-ID are unchanged in the new process image. Otherwise, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the new process image shall be set to the user ID of the new process image file. Similarly, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the new process image shall be set to the group ID of the new process image file. The real user ID, real group ID, and supplementary group IDs of the new process image shall remain the same as those of the calling process image. The effective user ID and effective group ID of the new process image shall be saved (as the saved set-user-ID and the saved set-group-ID) for use by *setuid*().

Any shared memory segments attached to the calling process image shall not be attached to the new process image.

Any named semaphores open in the calling process shall be closed as if by appropriate calls to *sem_close*().

Any blocks of typed memory that were mapped in the calling process are unmapped, as if *munmap*() was implicitly called to unmap them.

Memory locks established by the calling process via calls to *mlockall*() or *mlock*() shall be removed. If locked pages in the address space of the calling process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by the call by this process to the *exec* function. If the *exec* function fails, the effect on memory locks is unspecified.

Memory mappings created in the process are unmapped before the address space is rebuilt for the new process image.

When the calling process image does not use the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the scheduling policy and parameters of the new process image and the initial thread in that new process image are implementation-defined.

When the calling process image uses the SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC scheduling policies, the process policy and scheduling parameter settings shall not be changed by a call to an *exec* function. The initial thread in the new process image shall inherit the process scheduling policy and parameters. It shall have the default system contention scope, but shall inherit its allocation domain from the calling process image.

Per-process timers created by the calling process shall be deleted before replacing the current process image with the new process image.

All open message queue descriptors in the calling process shall be closed, as described in *mq_close*().

Any outstanding asynchronous I/O operations may be canceled. Those asynchronous I/O operations that are not canceled shall complete as if the *exec* function had not yet occurred, but any associated signal notifications shall be suppressed. It is unspecified whether the *exec* function itself blocks awaiting such I/O completion. In no event, however, shall the new process image created by the *exec* function be affected by the presence of outstanding asynchronous I/O operations at the time the *exec* function is called. Whether any I/O is canceled, and which I/O may be canceled upon *exec*, is implementation-defined.

The new process image shall inherit the CPU-time clock of the calling process image. This inheritance means that the process CPU-time clock of the process being *exec*-ed shall not be reinitialized or altered as a result of the *exec* function other than to reflect the time spent by the process executing the *exec* function itself.

The initial value of the CPU-time clock of the initial thread of the new process image shall be set to zero.

If the calling process is being traced, the new process image shall continue to be traced into the same trace stream as the original process image, but the new process image shall not inherit the mapping of trace event names to trace event type identifiers that was defined by calls to the *posix_trace_eventid_open*() or the *posix_trace_trid_eventid_open*() functions in the calling process image.

If the calling process is a trace controller process, any trace streams that were created by the calling process shall be shut down as described in the *posix_trace_shutdown*() function.

The thread ID of the initial thread in the new process image is unspecified.

The size and location of the stack on which the initial thread in the new process image runs is unspecified.

The initial thread in the new process image shall have its cancellation type set to PTHREAD_CANCEL_DEFERRED and its cancellation state set to PTHREAD_CANCEL_ENABLED.

The initial thread in the new process image shall have all thread-specific data values set to NULL and all thread-specific data keys shall be removed by the call to *exec* without running destructors.

The initial thread in the new process image shall be joinable, as if created with the *detachstate* attribute set to PTHREAD_CREATE_JOINABLE.

The new process shall inherit at least the following attributes from the calling process image:

* Nice value (see *nice*())
* *semadj* values (see *semop*())
* Process ID
* Parent process ID
* Process group ID
* Session membership
* Real user ID
* Real group ID
* Supplementary group IDs
* Time left until an alarm clock signal (see *alarm*())
* Current working directory
* Root directory
* File mode creation mask (see *umask*())
* File size limit (see *getrlimit*() and *setrlimit*())
* Process signal mask (see *pthread_sigmask*())
* Pending signal (see *sigpending*())
* *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* (see *times*())
* Resource limits
* Controlling terminal

* Interval timers

The initial thread of the new process shall inherit at least the following attributes from the calling thread:

* Signal mask (see *sigprocmask*() and *pthread_sigmask*())
* Pending signals (see *sigpending*())

All other process attributes defined in this volume of POSIX.1-2017 shall be inherited in the new process image from the old process image. All other thread attributes defined in this volume of POSIX.1-2017 shall be inherited in the initial thread in the new process image from the calling thread in the old process image. The inheritance of process or thread attributes not defined by this volume of POSIX.1-2017 is implementation-defined.

A call to any *exec* function from a process with more than one thread shall result in all threads being terminated and the new executable image being loaded and executed. No destructor functions or cleanup handlers shall be called.

Upon successful completion, the *exec* functions shall mark for update the last data access timestamp of the file. If an *exec* function failed but was able to locate the process image file, whether the last data access timestamp is marked for update is unspecified. Should the *exec* function succeed, the process image file shall be considered to have been opened with *open*(). The corresponding *close*() shall be considered to occur at a time after this open, but before process termination or successful completion of a subsequent call to one of the *exec* functions, *posix_spawn*(), or *posix_spawnp*(). The *argv*[ ] and *envp*[ ] arrays of pointers and the strings to which those arrays point shall not be modified by a call to one of the *exec* functions, except as a consequence of replacing the process image.

The saved resource limits in the new process image are set to be a copy of the process' corresponding hard and soft limits.

RETURN VALUE

If one of the *exec* functions returns to the calling process image, an error has occurred; the return value shall be −1, and *errno* shall be set to indicate the error.

ERRORS

The *exec* functions shall fail if:

E2BIG The number of bytes used by the new process image's argument list and environment list is greater than the system-imposed limit of {ARG_MAX} bytes.

EACCES
The new process image file is not a regular file and the implementation does not support execution of files of its type.

EINVAL
The new process image file has appropriate privileges and has a recognized executable binary format, but the system does not support execution of a file with this format.

The *exec* functions, except for *fexecve*(), shall fail if:

EACCES
Search permission is denied for a directory listed in the new process image file's path prefix, or the new process image file denies execution permission.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* or *file* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* or *file* does not name an existing file or *path* or *file* is an empty string.

ENOTDIR
A component of the new process image file's path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the new process image file's pathname contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *exec* functions, except for *execlp*() and *execvp*(), shall fail if:

ENOEXEC
The new process image file has the appropriate access permission but has an unrecognized format.

The *fexecve*() function shall fail if:

EBADF
The *fd* argument is not a valid file descriptor open for executing.

The *exec* functions may fail if:

ENOMEM
The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

The *exec* functions, except for *fexecve*(), may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* or *file* argument.

ENAMETOOLONG
The length of the *path* argument or the length of the pathname constructed from the *file* argument exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ETXTBSY
The new process image file is a pure procedure (shared text) file that is currently open for writing by some process.

The following sections are informative.

EXAMPLES

Using execl( )

The following example executes the *ls* command, specifying the pathname of the executable (**/bin/ls**) and using arguments supplied directly to the command to produce single-column output.

```
#include <unistd.h>

int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

Using execle( )

The following example is similar to *Using execl( )*. In addition, it specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execle ("/bin/ls", "ls", "-l", (char *)0, env);
```

Using execlp( )

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable.

```
#include <unistd.h>

int ret;
...
ret = execlp ("ls", "ls", "-l", (char *)0);
```

Using execv( )

The following example passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execv ("/bin/ls", cmd);
```

Using execve( )

The following example passes arguments to the *ls* command in the *cmd* array, and specifies the environment for the new process image using the *env* argument.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Using execvp( )

The following example searches for the location of the *ls* command among the directories specified by the *PATH* environment variable, and passes arguments to the *ls* command in the *cmd* array.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

APPLICATION USAGE

As the state of conversion descriptors and message catalog descriptors in the new process image is undefined, conforming applications should not rely on their use and should close them prior to calling one of the *exec* functions.

Applications that require other than the default POSIX locale as the global locale in the new process image should call *setlocale*() with the appropriate parameters.

When assigning a new value to the *environ* variable, applications should ensure that the environment to which it will point contains at least the following:

1. Any implementation-defined variables required by the implementation to provide a conforming environment. See the _CS_V7_ENV entry in *<unistd.h>* and *confstr*() for details.

2. A value for *PATH* which finds conforming versions of all standard utilities before any other versions.

The same constraint applies to the *envp* array passed to *execle*() or *execve*(), in order to ensure that the new process image is invoked in a conforming environment.

Applications should not execute programs with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, as this might cause the executed program to misbehave. In order not to pass on these file descriptors to an executed program, applications should not just close them but should reopen them on, for example, **/dev/null**. Some implementations may reopen them automatically, but applications should not rely on this being done.

If an application wants to perform a checksum test of the file being executed before executing it, the file will need to be opened with read permission to perform the checksum test.

Since execute permission is checked by *fexecve*(), the file description *fd* need not have been opened with the O_EXEC flag. However, if the file to be executed denies read and write permission for the process preparing to do the *exec*, the only way to provide the *fd* to *fexecve*() will be to use the O_EXEC flag when opening *fd*. In this case, the application will not be able to perform a checksum test since it will not be able to read the contents of the file.

Note that when a file descriptor is opened with O_RDONLY, O_RDWR, or O_WRONLY mode, the file descriptor can be used to read, read and write, or write the file, respectively, even if the mode of the file changes after the file was opened. Using the O_EXEC open mode is different; *fexecve*() will ignore the mode that was used when the file descriptor was opened and the *exec* will fail if the mode of the file associated with *fd* does not grant execute permission to the calling process at the time *fexecve*() is called.

RATIONALE

Early proposals required that the value of *argc* passed to *main*() be "one or greater". This was driven by the same requirement in drafts of the ISO C standard. In fact, historical implementations have passed a value of zero when no arguments are supplied to the caller of the *exec* functions. This requirement was removed from the ISO C standard and subsequently removed from this volume of POSIX.1-2017 as well. The wording, in particular the use of the word *should*, requires a Strictly Conforming POSIX Application to pass at least one argument to the *exec* function, thus guaranteeing that *argc* be one or greater when invoked by such an application. In fact, this is good practice, since many existing applications reference *argv*[0] without first checking the value of *argc*.

The requirement on a Strictly Conforming POSIX Application also states that the value passed as the first argument be a filename string associated with the process being started. Although some existing applications pass a pathname rather than a filename string in some circumstances, a filename string is more generally useful, since the common usage of *argv*[0] is in printing diagnostics. In some cases the filename passed is not the actual filename of the file; for example, many implementations of the *login* utility use a convention of prefixing a <hyphen-minus> (**'-'**) to the actual filename, which indicates to the command interpreter being invoked that it is a "login shell".

Also, note that the *test* and *[* utilities require specific strings for the *argv*[0] argument to have deterministic behavior across all implementations.

Historically, there have been two ways that implementations can *exec* shell scripts.

One common historical implementation is that the *execl*(), *execv*(), *execle*(), and *execve*() functions return an **[ENOEXEC]** error for any file not recognizable as executable, including a shell script. When the *execlp*() and *execvp*() functions encounter such a file, they assume the file to be a shell script and invoke a known command interpreter to interpret such files. This is now required by POSIX.1-2008. These implementations of *execvp*() and *execlp*() only give the **[ENOEXEC]** error in the rare case of a problem with the command interpreter's executable file. Because of these implementations, the **[ENOEXEC]** error is not mentioned for *execlp*() or *execvp*(), although implementations can still give it.

Another way that some historical implementations handle shell scripts is by recognizing the first two bytes of the file as the character string **"#!"** and using the remainder of the first line of the file as the name of the command interpreter to execute.

One potential source of confusion noted by the standard developers is over how the contents of a process image file affect the behavior of the *exec* family of functions. The following is a description of the actions taken:

1. If the process image file is a valid executable (in a format that is executable and valid and having appropriate privileges) for this system, then the system executes the file.
2. If the process image file has appropriate privileges and is in a format that is executable but not valid for this system (such as a recognized binary for another architecture), then this is an error and *errno* is set to **[EINVAL]** (see later RATIONALE on **[EINVAL]**).
3. If the process image file has appropriate privileges but is not otherwise recognized:
   a. If this is a call to *execlp*() or *execvp*(), then they invoke a command interpreter assuming that the process image file is a shell script.
   b. If this is not a call to *execlp*() or *execvp*(), then an error occurs and *errno* is set to **[ENOEXEC]**.

Applications that do not require to access their arguments may use the form:

```
main(void)
```

as specified in the ISO C standard. However, the implementation will always provide the two arguments *argc* and *argv*, even if they are not used.

Some implementations provide a third argument to *main*() called *envp*. This is defined as a pointer to the environment. The ISO C standard specifies invoking *main*() with two arguments, so implementations must support applications written this way. Since this volume of POSIX.1-2017 defines the global variable *environ*, which is also provided by historical implementations and can be used anywhere that *envp* could be used, there is no functional need for the *envp* argument. Applications should use the *getenv*() function rather than accessing the environment directly via either *envp* or *environ*. Implementations are required to support the two-argument calling sequence, but this does not prohibit an implementation from supporting *envp* as an optional third argument.

This volume of POSIX.1-2017 specifies that signals set to SIG_IGN remain set to SIG_IGN, and that the new process image inherits the signal mask of the thread that called *exec* in the old process image. This is consistent with historical implementations, and it permits some useful functionality, such as the *nohup* command. However, it should be noted that many existing applications wrongly assume that they start with certain signals set to the default action and/or unblocked. In particular, applications written with a simpler signal model that does not include blocking of signals, such as the one in the ISO C standard, may not behave properly if invoked with some signals blocked. Therefore, it is best not to block or ignore signals across *exec*s without explicit reason to do so, and especially not to block signals across *exec*s of arbitrary (not closely cooperating) programs.

The *exec* functions always save the value of the effective user ID and effective group ID of the process at the completion of the *exec*, whether or not the set-user-ID or the set-group-ID bit of the process image file is set.

The statement about *argv*[ ] and *envp*[ ] being constants is included to make explicit to future writers of language bindings that these objects are completely constant. Due to a limitation of the ISO C standard, it is not possible to state that idea in standard C. Specifying two levels of *const−qualification* for the *argv*[ ] and *envp*[ ] parameters for the *exec* functions may seem to be the natural choice, given that these functions do not modify either the array of pointers or the characters to which the function points, but this would disallow existing correct code. Instead, only the array of pointers is noted as constant. The table of assignment compatibility for *dst*=*src* derived from the ISO C standard summarizes the compatibility:

box tab(!) center; r | lB | lB | lB | lB lB | c | c | c | c. *dst*:!char *[ ]!const char *[ ]!char *const[ ]!const char *const[ ] _ *src*: char *[ ]!VALID!—!VALID!— const char *[ ]!—!VALID!—!VALID char * const [ ]!—!—!VALID!— const char *const[ ]!—!—!—!VALID

Since all existing code has a source type matching the first row, the column that gives the most valid combinations is the third column. The only other possibility is the fourth column, but using it would require a cast on the *argv* or *envp* arguments. It is unfortunate that the fourth column cannot be used, because the declaration a non-expert would naturally use would be that in the second row.

The ISO C standard and this volume of POSIX.1-2017 do not conflict on the use of *environ*, but some historical implementations of *environ* may cause a conflict. As long as *environ* is treated in the same way as an entry point (for example, *fork*()), it conforms to both standards. A library can contain *fork*(), but if there is a user-provided *fork*(), that *fork*() is given precedence and no problem ensues. The situation is similar for *environ*: the definition in this volume of POSIX.1-2017 is to be used if there is no user-provided *environ* to take precedence. At least three implementations are known to exist that solve this problem.

E2BIG The limit {ARG_MAX} applies not just to the size of the argument list, but to the sum of that and the size of the environment list.

EFAULT
Some historical systems return **[EFAULT]** rather than **[ENOEXEC]** when the new process image file is corrupted. They are non-conforming.

EINVAL
This error condition was added to POSIX.1-2008 to allow an implementation to detect executable files generated for different architectures, and indicate this situation to the application. Historical implementations of shells, *execvp*(), and *execlp*() that encounter an **[ENOEXEC]** error will execute a shell on the assumption that the file is a shell script. This will not produce the desired effect when the file is a valid executable for a different architecture. An implementation may now choose to avoid this problem by returning **[EINVAL]** when a valid executable for a different architecture is encountered. Some historical implementations return **[EINVAL]** to indicate that the *path* argument contains a character with the high order bit set. The standard developers chose to deviate from historical practice for the following reasons:

1. The new utilization of **[EINVAL]** will provide some measure of utility to the user community.
2. Historical use of **[EINVAL]** is not acceptable in an internationalized operating environment.

ENAMETOOLONG
Since the file pathname may be constructed by taking elements in the *PATH* variable and putting them together with the filename, the **[ENAMETOOLONG]** error condition could also be reached this way.

ETXTBSY
System V returns this error when the executable file is currently open for writing by some process. This volume of POSIX.1-2017 neither requires nor prohibits this behavior.

Other systems (such as System V) may return **[EINTR]** from *exec*. This is not addressed by this volume of POSIX.1-2017, but implementations may have a window between the call to *exec* and the time that a signal could cause one of the *exec* calls to return with **[EINTR]**.

An explicit statement regarding the floating-point environment (as defined in the <*fenv.h*> header) was added to make it clear that the floating-point environment is set to its default when a call to one of the *exec* functions succeeds. The requirements for inheritance or setting to the default for other process and thread start-up functions is covered by more generic statements in their descriptions and can be summarized as follows:

posix_spawn( )	Set to default.
fork( )	Inherit.
pthread_create( )	Inherit.

The purpose of the *fexecve*() function is to enable executing a file which has been verified to be the intended file. It is possible to actively check the file by reading from the file descriptor and be sure that the file is not exchanged for another between the reading and the execution. Alternatively, a function like *openat*() can be used to open a file which has been found by reading the content of a directory using *readdir*().

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *atexit*( ), *chmod*( ), *close*( ), *confstr*( ), *exit*( ), *fcntl*( ), *fork*( ), *fstatvfs*( ), *getenv*( ), *getitimer*( ), *getrlimit*( ), *mknod*( ), *mmap*( ), *nice*( ), *open*( ), *posix_spawn*( ), *posix_trace_create*( ), *posix_trace_event*( ), *posix_trace_eventid_equal*( ), *pthread_atfork*( ), *pthread_sigmask*( ), *putenv*( ), *readdir*( ), *semop*( ), *setlocale*( ), *shmat*( ), *sigaction*( ), *sigaltstack*( ), *sigpending*( ), *system*( ), *times*( ), *ulimit*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*, **<unistd.h>**

The Shell and Utilities volume of POSIX.1-2017, *test*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

exit — terminate a process

SYNOPSIS

```
#include <stdlib.h>

void exit(int status);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The value of *status* may be 0, EXIT_SUCCESS, EXIT_FAILURE, or any other value, though only the least significant 8 bits (that is, *status* & 0377) shall be available from *wait*() and *waitpid*(); the full value shall be available from *waitid*() and in the **siginfo_t** passed to a signal handler for SIGCHLD.

The *exit*() function shall first call all functions registered by *atexit*(), in the reverse order of their registration, except that a function is called after any previously registered functions that had already been called at the time it was registered. Each function is called as many times as it was registered. If, during the call to any such function, a call to the *longjmp*() function is made that would terminate the call to the registered function, the behavior is undefined.

If a function registered by a call to *atexit*() fails to return, the remaining registered functions shall not be called and the rest of the *exit*() processing shall not be completed. If *exit*() is called more than once, the behavior is undefined.

The *exit*() function shall then flush all open streams with unwritten buffered data and close all open streams. Finally, the process shall be terminated with the same consequences as described in *Consequences of Process Termination*.

RETURN VALUE

The *exit*() function does not return.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See *_Exit*().

FUTURE DIRECTIONS

None.

SEE ALSO

_Exit( ), *atexit*( ), *exec*, *longjmp*( ), *tmpfile*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

exp, expf, expl — exponential function

SYNOPSIS

```
#include <math.h>

double exp(double x);
float expf(float x);
long double expl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the base-*e* exponential of *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponential value of *x*.

If the correct value would cause overflow, a range error shall occur and *exp*(), *expf*(), and *expl*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If the correct value would cause underflow, and is not representable, a range error may occur, and *exp*(), *expf*(), and *expl*() shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, 1 shall be returned.

If *x* is −Inf, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Computing the Density of the Standard Normal Distribution

This function shows an implementation for the density of the standard normal distribution using *exp*(). This example uses the constant M_PI which is part of the XSI option.

```
#include <math.h>

double
normal_density (double x)
{
   return exp(-x*x/2) / sqrt (2*M_PI);
}
```

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

exp2, exp2f, exp2l — exponential base 2 functions

SYNOPSIS

#include <math.h>

double exp2(double *x*);
float exp2f(float *x*);
long double exp2l(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the base-2 exponential of *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return 2^x.

If the correct value would cause overflow, a range error shall occur and *exp2*(), *exp2f*(), and *exp2l*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If the correct value would cause underflow, and is not representable, a range error may occur, and *exp2*(), *exp2f*(), and *exp2l*() shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, 1 shall be returned.

If *x* is −Inf, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

expm1, expm1f, expm1l — compute exponential functions

SYNOPSIS

```
#include <math.h>

double expm1(double x);
float expm1f(float x);
long double expm1l(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute $e^x - 1.0$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions return $e^x - 1.0$.

If the correct value would cause overflow, a range error shall occur and *expm1*(), *expm1f*(), and *expm1l*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is −Inf, −1 shall be returned.

If *x* is +Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *expm1*(), *expm1f*(), and *expm1l*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The value of *expm1*(*x*) may be more accurate than *exp*(*x*)−1.0 for small values of *x*.

The *expm1*() and *log1p*() functions are useful for financial calculations of $((1+x)^n-1)/x$, namely:

```
expm1(n * log1p(x))/x
```

when *x* is very small (for example, when calculating small daily interest rates). These functions also simplify writing accurate inverse hyperbolic functions.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *ilogb*( ), *log1p*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fabs, fabsf, fabsl — absolute value function

SYNOPSIS

```
#include <math.h>

double fabs(double x);
float fabsf(float x);
long double fabsl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the absolute value of their argument x,$|x|$.

RETURN VALUE

Upon successful completion, these functions shall return the absolute value of x.

If x is NaN, a NaN shall be returned.

If x is ±0, +0 shall be returned.

If x is ±Inf, +Inf shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Computing the 1-Norm of a Floating-Point Vector

This example shows the use of *fabs*() to compute the 1-norm of a vector defined as follows:

$$\text{norm1}(v) = |v[0]| + |v[1]| + \ldots + |v[n-1]|$$

where $|x|$ denotes the absolute value of x, n denotes the vector's dimension $v[i]$ denotes the i-th component of v ($0 \le i < n$).

```
#include <math.h>

double
norm1(const double v[], const int n)
{
    int     i;
    double  n1_v;  /* 1-norm of v */

    n1_v = 0;
    for (i=0; i<n; i++) {
        n1_v += fabs (v[i]);
    }

    return n1_v;
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

faccessat — determine accessibility of a file relative to directory file descriptor

SYNOPSIS

#include <unistd.h>

int faccessat(int *fd*, const char **path*, int *amode*, int *flag*);

DESCRIPTION

Refer to *access*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fattach — attach a STREAMS-based file descriptor to a file in the file system name space (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int fattach(int fildes, const char *path);
```

DESCRIPTION

The *fattach*() function shall attach a STREAMS-based file descriptor to a file, effectively associating a pathname with *fildes*. The application shall ensure that the *fildes* argument is a valid open file descriptor associated with a STREAMS file. The *path* argument points to a pathname of an existing file. The application shall have appropriate privileges or be the owner of the file named by *path* and have write permission. A successful call to *fattach*() shall cause all pathnames that name the file named by *path* to name the STREAMS file associated with *fildes*, until the STREAMS file is detached from the file. A STREAMS file can be attached to more than one file and can have several pathnames associated with it.

The attributes of the named STREAMS file shall be initialized as follows: the permissions, user ID, group ID, and times are set to those of the file named by *path*, the number of links is set to 1, and the size and device identifier are set to those of the STREAMS file associated with *fildes*. If any attributes of the named STREAMS file are subsequently changed (for example, by *chmod*()), neither the attributes of the underlying file nor the attributes of the STREAMS file to which *fildes* refers shall be affected.

File descriptors referring to the underlying file, opened prior to an *fattach*() call, shall continue to refer to the underlying file.

RETURN VALUE

Upon successful completion, *fattach*() shall return 0. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fattach*() function shall fail if:

EACCES
Search permission is denied for a component of the path prefix, or the process is the owner of *path* but does not have write permissions on the file named by *path*.

EBADF
The *fildes* argument is not a valid open file descriptor.

EBUSY
The file named by *path* is currently a mount point or has a STREAMS file attached to it.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters.

EPERM
The effective user ID of the process is not the owner of the file named by *path* and the process does not have appropriate privileges.

The *fattach*() function may fail if:

EINVAL
The *fildes* argument does not refer to a STREAMS file.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EXDEV
A link to a file on another file system was attempted.

The following sections are informative.

EXAMPLES

Attaching a File Descriptor to a File

In the following example, *fd* refers to an open STREAMS file. The call to *fattach*() associates this STREAM with the file **/tmp/named-STREAM**, such that any future calls to open **/tmp/named-STREAM**, prior to breaking the attachment via a call to *fdetach*(), will instead create a new file handle referring to the STREAMS file associated with *fd*.

```
#include <stropts.h>
...
  int fd;
  char *pathname = "/tmp/named-STREAM";
  int ret;

  ret = fattach(fd, pathname);
```

APPLICATION USAGE

The *fattach*() function behaves similarly to the traditional *mount*() function in the way a file is temporarily replaced by the root directory of the mounted file system. In the case of *fattach*(), the replaced file need not be a directory and the replacing file is a STREAMS file.

RATIONALE

The file attributes of a file which has been the subject of an *fattach*() call are specifically set because of an artifact of the original implementation. The internal mechanism was the same as for the *mount*() function. Since *mount*() is typically only applied to directories, the effects when applied to a regular file are a little surprising, especially as regards the link count which rigidly remains one, even if there were several links originally and despite the fact that all original links refer to the STREAM as long as the *fattach*() remains in effect.

FUTURE DIRECTIONS

The *fattach*() function may be removed in a future version.

SEE ALSO

fdetach( ), *isastream*( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fchdir — change working directory

SYNOPSIS

#include <unistd.h>

int fchdir(int *fildes*);

DESCRIPTION

The *fchdir*() function shall be equivalent to *chdir*() except that the directory that is to be the new current working directory is specified by the file descriptor *fildes*.

A conforming application can obtain a file descriptor for a file of type directory using *open*(), provided that the file status flags and access modes do not contain O_WRONLY or O_RDWR.

RETURN VALUE

Upon successful completion, *fchdir*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error. On failure the current working directory shall remain unchanged.

ERRORS

The *fchdir*() function shall fail if:

EACCES
Search permission is denied for the directory referenced by *fildes*.

EBADF
The *fildes* argument is not an open file descriptor.

ENOTDIR
The open file descriptor *fildes* does not refer to a directory.

The *fchdir*() may fail if:

EINTR
A signal was caught during the execution of *fchdir*().

EIO An I/O error occurred while reading from or writing to the file system.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chdir( ), *dirfd*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fchmod — change mode of a file

SYNOPSIS

#include <sys/stat.h>

int fchmod(int *fildes*, mode_t *mode*);

DESCRIPTION

The *fchmod*() function shall be equivalent to *chmod*() except that the file whose permissions are changed is specified by the file descriptor *fildes*.

If *fildes* references a shared memory object, the *fchmod*() function need only affect the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits.

If *fildes* references a typed memory object, the behavior of *fchmod*() is unspecified.

If *fildes* refers to a socket, the behavior of *fchmod*() is unspecified.

If *fildes* refers to a STREAM (which is *fattach*()-ed into the file system name space) the call returns successfully, doing nothing.

RETURN VALUE

Upon successful completion, *fchmod*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *fchmod*() function shall fail if:

EBADF
The *fildes* argument is not an open file descriptor.

EPERM
The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

EROFS
The file referred to by *fildes* resides on a read-only file system.

The *fchmod*() function may fail if:

EINTR
The *fchmod*() function was interrupted by a signal.

EINVAL
The value of the *mode* argument is invalid.

EINVAL
The *fildes* argument refers to a pipe and the implementation disallows execution of *fchmod*() on a pipe.

The following sections are informative.

EXAMPLES

Changing the Current Permissions for a File

The following example shows how to change the permissions for a file named **/home/cnd/mod1** so that the owner and group have read/write/execute permissions, but the world only has read/write permissions.

```
#include <sys/stat.h>
#include <fcntl.h>
```

```
mode_t mode;
int    fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
fchmod(fildes, S_IRWXU | S_IRWXG | S_IROTH | S_IWOTH);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *fcntl*( ), *fstatat*( ), *fstatvfs*( ), *mknod*( ), *open*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_stat.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fchmodat — change mode of a file relative to directory file descriptor

SYNOPSIS

#include <sys/stat.h>

int fchmodat(int *fd*, const char **path*, mode_t *mode*, int *flag*);

DESCRIPTION

Refer to *chmod*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fchown — change owner and group of a file

SYNOPSIS

#include <unistd.h>

int fchown(int *fildes*, uid_t *owner*, gid_t *group*);

DESCRIPTION

The *fchown*() function shall be equivalent to *chown*() except that the file whose owner and group are changed is specified by the file descriptor *fildes*.

RETURN VALUE

Upon successful completion, *fchown*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *fchown*() function shall fail if:

EBADF
The *fildes* argument is not an open file descriptor.

EPERM
The effective user ID does not match the owner of the file or the process does not have appropriate privileges and _POSIX_CHOWN_RESTRICTED indicates that such privilege is required.

EROFS
The file referred to by *fildes* resides on a read-only file system.

The *fchown*() function may fail if:

EINVAL
The owner or group ID is not a value supported by the implementation. The *fildes* argument refers to a pipe or socket or an *fattach*()-ed STREAM and the implementation disallows execution of *fchown*() on a pipe.

EIO A physical I/O error has occurred.

EINTR
The *fchown*() function was interrupted by a signal which was caught.

The following sections are informative.

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the owner of a file named **/home/cnd/mod1** to "jones" and the group to "cnd".

The numeric value for the user ID is obtained by extracting the user ID from the user database entry associated with "jones". Similarly, the numeric value for the group ID is obtained by extracting the group ID from the group database entry associated with "cnd". This example assumes the calling program has appropriate privileges.

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <pwd.h>
#include <grp.h>
```

```
struct passwd *pwd;
struct group  *grp;
int           fildes;
...
fildes = open("/home/cnd/mod1", O_RDWR);
pwd = getpwnam("jones");
grp = getgrnam("cnd");
fchown(fildes, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fchownat — change owner and group of a file relative to directory file descriptor

SYNOPSIS

```
#include <unistd.h>

int fchownat(int fd, const char *path, uid_t owner, gid_t group,
    int flag);
```

DESCRIPTION

Refer to *chown*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fclose — close a stream

SYNOPSIS

#include <stdio.h>

int fclose(FILE ***stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fclose*() function shall cause the stream pointed to by *stream* to be flushed and the associated file to be closed. Any unwritten buffered data for the stream shall be written to the file; any unread buffered data shall be discarded. Whether or not the call succeeds, the stream shall be disassociated from the file and any buffer set by the *setbuf*() or *setvbuf*() function shall be disassociated from the stream. If the associated buffer was automatically allocated, it shall be deallocated.

If the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description shall be set to the file position of the stream if the stream is the active handle to the underlying file description.

The *fclose*() function shall mark for update the last data modification and last file status change timestamps of the underlying file, if the stream was writable, and if buffered data remains that has not yet been written to the file. The *fclose*() function shall perform the equivalent of a *close*() on the file descriptor that is associated with the stream pointed to by *stream*.

After the call to *fclose*(), any use of *stream* results in undefined behavior.

RETURN VALUE

Upon successful completion, *fclose*() shall return 0; otherwise, it shall return EOF and set *errno* to indicate the error.

ERRORS

The *fclose*() function shall fail if:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying stream is not valid.

EFBIG
An attempt was made to write a file that exceeds the maximum file size.

EFBIG
An attempt was made to write a file that exceeds the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR
The *fclose*() function was interrupted by a signal.

EIO The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOMEM
The underlying stream was created by *open_memstream*() or *open_wmemstream*() and insufficient memory is available.

ENOSPC
There was no free space remaining on the device containing the file or in the buffer used by the *fmemopen*() function.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fclose*() function may fail if:

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since after the call to *fclose*() any use of *stream* results in undefined behavior, *fclose*() should not be used on *stdin*, *stdout*, or *stderr* except immediately before process termination (see the Base Definitions volume of POSIX.1-2017, *Section 3.303*, *Process Termination*), so as to avoid triggering undefined behavior in other standard interfaces that rely on these streams. If there are any *atexit*() handlers registered by the application, such a call to *fclose*() should not occur until the last handler is finishing. Once *fclose*() has been used to close *stdin*, *stdout*, or *stderr*, there is no standard way to reopen any of these streams.

Use of *freopen*() to change *stdin*, *stdout*, or *stderr* instead of closing them avoids the danger of a file unexpectedly being opened as one of the special file descriptors STDIN_FILENO, STDOUT_FILENO, or STDERR_FILENO at a later time in the application.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *atexit*( ), *close*( ), *fmemopen*( ), *fopen*( ), *freopen*( ), *getrlimit*( ), *open_memstream*( ), *ulimit*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fcntl — file control

SYNOPSIS

#include <fcntl.h>

int fcntl(int *fildes*, int *cmd*, ...);

DESCRIPTION

The *fcntl*() function shall perform the operations described below on open files. The *fildes* argument is a file descriptor.

The available values for *cmd* are defined in <*fcntl.h*> and are as follows:

F_DUPFD Return a new file descriptor which shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*, except that it shall be the lowest numbered available file descriptor greater than or equal to the third argument, *arg*, taken as an integer of type **int**. The new file descriptor shall refer to the same open file description as the original file descriptor, and shall share any locks. The FD_CLOEXEC flag associated with the new file descriptor shall be cleared to keep the file open across calls to one of the *exec* functions.

F_DUPFD_CLOEXEC
Like F_DUPFD, but the FD_CLOEXEC flag associated with the new file descriptor shall be set.

F_GETFD Get the file descriptor flags defined in <*fcntl.h*> that are associated with the file descriptor *fildes*. File descriptor flags are associated with a single file descriptor and do not affect other file descriptors that refer to the same file.

F_SETFD Set the file descriptor flags defined in <*fcntl.h*>, that are associated with *fildes*, to the third argument, *arg*, taken as type **int**. If the FD_CLOEXEC flag in the third argument is 0, the file descriptor shall remain open across the *exec* functions; otherwise, the file descriptor shall be closed upon successful execution of one of the *exec* functions.

F_GETFL Get the file status flags and file access modes, defined in <*fcntl.h*>, for the file description associated with *fildes*. The file access modes can be extracted from the return value using the mask O_ACCMODE, which is defined in <*fcntl.h*>. File status flags and file access modes are associated with the file description and do not affect other file descriptors that refer to the same file with different open file descriptions. The flags returned may include non-standard file status flags which the application did not set, provided that these additional flags do not alter the behavior of a conforming application.

F_SETFL Set the file status flags, defined in <*fcntl.h*>, for the file description associated with *fildes* from the corresponding bits in the third argument, *arg*, taken as type **int**. Bits corresponding to the file access mode and the file creation flags, as defined in <*fcntl.h*>, that are set in *arg* shall be ignored. If any bits in *arg* other than those mentioned here are changed by the application, the result is unspecified. If *fildes* does not support non-blocking operations, it is unspecified whether the O_NONBLOCK flag will be ignored.

F_GETOWN If *fildes* refers to a socket, get the process ID or process group ID specified to receive SIGURG signals when out-of-band data is available. Positive values shall indicate a process ID; negative values, other than −1, shall indicate a process group ID; the value zero shall indicate that no SIGURG signals are to be sent. If *fildes* does not refer to a socket, the results are unspecified.

F_SETOWN If *fildes* refers to a socket, set the process ID or process group ID specified to receive SIGURG signals when out-of-band data is available, using the value of the third argument, *arg*, taken as type **int**. Positive values shall indicate a process ID; negative values, other than −1, shall indicate a process group ID; the value zero shall indicate that no SIGURG signals are to be sent. Each time a SIGURG signal is sent to the specified process or process group, permission checks equivalent to those performed by *kill*() shall be performed, as if *kill*() were called by a process with the same real user ID, effective user ID, and privileges that the process calling *fcntl*() has at the time of the call; if the *kill*() call would fail, no signal shall be sent. These permission checks may also be performed by the *fcntl*() call. If the process specified by *arg* later terminates, or the process group specified by *arg* later becomes empty, while still being specified to receive SIGURG signals when out-of-band data is available from *fildes*, then no signals shall be sent to any subsequently created process that has the same process ID or process group ID, regardless of permission; it is unspecified whether this is achieved by the equivalent of a *fcntl*(*fildes*, F_SETOWN, 0) call at the time the process terminates or is waited for or the process group becomes empty, or by other means. If *fildes* does not refer to a socket, the results are unspecified.

The following values for *cmd* are available for advisory record locking. Record locking shall be supported for regular files, and may be supported for other files.

F_GETLK Get any lock which blocks the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in <*fcntl.h*>. The information retrieved shall overwrite the information passed to *fcntl*() in the structure **flock**. If no lock is found that would prevent this lock from being created, then the structure shall be left unchanged except for the lock type which shall be set to F_UNLCK.

F_SETLK Set or clear a file segment lock according to the lock description pointed to by the third argument, *arg*, taken as a pointer to type **struct flock**, defined in <*fcntl.h*>. F_SETLK can establish shared (or read) locks (F_RDLCK) or exclusive (or write) locks (F_WRLCK), as well as to remove either type of lock (F_UNLCK). F_RDLCK, F_WRLCK, and F_UNLCK are defined in <*fcntl.h*>. If a shared or exclusive lock cannot be set, *fcntl*() shall return immediately with a return value of −1.

F_SETLKW This command shall be equivalent to F_SETLK except that if a shared or exclusive lock is blocked by other locks, the thread shall wait until the request can be satisfied. If a signal that is to be caught is received while *fcntl*() is waiting for a region, *fcntl*() shall be interrupted. Upon return from the signal handler, *fcntl*() shall return −1 with *errno* set to **[EINTR]**, and the lock operation shall not be done.

Additional implementation-defined values for *cmd* may be defined in <*fcntl.h*>. Their names shall start with F_.

When a shared lock is set on a segment of a file, other processes shall be able to set shared locks on that segment or a portion of it. A shared lock prevents any other process from setting an exclusive lock on any portion of the protected area. A request for a shared lock shall fail if the file descriptor was not opened with read access.

An exclusive lock shall prevent any other process from setting a shared lock or an exclusive lock on any portion of the protected area. A request for an exclusive lock shall fail if the file descriptor was not opened with write access.

The structure **flock** describes the type (*l_type*), starting offset (*l_whence*), relative offset (*l_start*), size (*l_len*), and process ID (*l_pid*) of the segment of the file to be affected.

The value of *l_whence* is SEEK_SET, SEEK_CUR, or SEEK_END, to indicate that the relative offset *l_start* bytes shall be measured from the start of the file, current position, or end of the file, respectively. The value of *l_len* is the number of consecutive bytes to be locked. The value of *l_len* may be negative (where the definition of **off_t** permits negative values of *l_len*). The *l_pid* field is only used with F_GETLK to return the process ID of the process holding a blocking lock. After a successful F_GETLK request, when a blocking lock is found, the values returned in the **flock** structure shall be as follows:

l_type	Type of blocking lock found.
l_whence	SEEK_SET.
l_start	Start of the blocking lock.
l_len	Length of the blocking lock.
l_pid	Process ID of the process that holds the blocking lock.

If the command is F_SETLKW and the process must wait for another process to release a lock, then the range of bytes to be locked shall be determined before the *fcntl*() function blocks. If the file size or file descriptor seek offset change while *fcntl*() is blocked, this shall not affect the range of bytes locked.

If *l_len* is positive, the area affected shall start at *l_start* and end at *l_start*+*l_len*−1. If *l_len* is negative, the area affected shall start at *l_start*+*l_len* and end at *l_start*−1. Locks may start and extend beyond the current end of a file, but shall not extend before the beginning of the file. A lock shall be set to extend to the largest possible value of the file offset for that file by setting *l_len* to 0. If such a lock also has *l_start* set to 0 and *l_whence* is set to SEEK_SET, the whole file shall be locked.

There shall be at most one type of lock set for each byte in the file. Before a successful return from an F_SETLK or an F_SETLKW request when the calling process has previously existing locks on bytes in the region specified by the request, the previous lock type for each byte in the specified region shall be replaced by the new lock type. As specified above under the descriptions of shared locks and exclusive locks, an F_SETLK or an F_SETLKW request (respectively) shall fail or block when another process has existing locks on bytes in the specified region and the type of any of those locks conflicts with the type specified in the request.

All locks associated with a file for a given process shall be removed when a file descriptor for that file is closed by that process or the process holding that file descriptor terminates. Locks are not inherited by a child process.

A potential for deadlock occurs if a process controlling a locked region is put to sleep by attempting to lock the locked region of another process. If the system detects that sleeping until a locked region is unlocked would cause a deadlock, *fcntl*() shall fail with an **[EDEADLK]** error.

An unlock (F_UNLCK) request in which *l_len* is non-zero and the offset of the last byte of the requested segment is the maximum value for an object of type **off_t**, when the process has an existing lock in which *l_len* is 0 and which includes the last byte of the requested segment, shall be treated as a request to unlock from the start of the requested segment with an *l_len* equal to 0. Otherwise, an unlock (F_UNLCK) request shall attempt to unlock only the requested segment.

When the file descriptor *fildes* refers to a shared memory object, the behavior of *fcntl*() shall be the same as for a regular file except the effect of the following values for the argument *cmd* shall be unspecified: F_SETFL, F_GETLK, F_SETLK, and F_SETLKW.

If *fildes* refers to a typed memory object, the result of the *fcntl*() function is unspecified.

RETURN VALUE

Upon successful completion, the value returned shall depend on *cmd* as follows:

F_DUPFD	A new file descriptor.
F_DUPFD_CLOEXEC	A new file descriptor.
F_GETFD	Value of flags defined in <*fcntl.h*>. The return value shall not be negative.
F_SETFD	Value other than −1.
F_GETFL	Value of file status flags and access modes. The return value is not negative.
F_SETFL	Value other than −1.
F_GETLK	Value other than −1.

F_SETLK	Value other than −1.
F_SETLKW	Value other than −1.
F_GETOWN	Value of the socket owner process or process group; this will not be −1.
F_SETOWN	Value other than −1.

Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fcntl*() function shall fail if:

EACCES or **EAGAIN**
The *cmd* argument is F_SETLK; the type of lock (*l_type*) is a shared (F_RDLCK) or exclusive (F_WRLCK) lock and the segment of a file to be locked is already exclusive-locked by another process, or the type is an exclusive lock and some portion of the segment of a file to be locked is already shared-locked or exclusive-locked by another process.

EBADF
The *fildes* argument is not a valid open file descriptor, or the argument *cmd* is F_SETLK or F_SETLKW, the type of lock, *l_type*, is a shared lock (F_RDLCK), and *fildes* is not a valid file descriptor open for reading, or the type of lock, *l_type*, is an exclusive lock (F_WRLCK), and *fildes* is not a valid file descriptor open for writing.

EINTR
The *cmd* argument is F_SETLKW and the function was interrupted by a signal.

EINVAL
The *cmd* argument is invalid, or the *cmd* argument is F_DUPFD or F_DUPFD_CLOEXEC and *arg* is negative or greater than or equal to {OPEN_MAX}, or the *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the data pointed to by *arg* is not valid, or *fildes* refers to a file that does not support locking.

EMFILE
The argument *cmd* is F_DUPFD or F_DUPFD_CLOEXEC and all file descriptors available to the process are currently open, or no file descriptors greater than or equal to *arg* are available.

ENOLCK
The argument *cmd* is F_SETLK or F_SETLKW and satisfying the lock or unlock request would result in the number of locked regions in the system exceeding a system-imposed limit.

EOVERFLOW
One of the values to be returned cannot be represented correctly.

EOVERFLOW
The *cmd* argument is F_GETLK, F_SETLK, or F_SETLKW and the smallest or, if *l_len* is non-zero, the largest offset of any byte in the requested segment cannot be represented correctly in an object of type **off_t**.

ESRCH
The *cmd* argument is F_SETOWN and no process or process group can be found corresponding to that specified by *arg*.

The *fcntl*() function may fail if:

EDEADLK
The *cmd* argument is F_SETLKW, the lock is blocked by a lock from another process, and putting the calling process to sleep to wait for that lock to become free would cause a deadlock.

EINVAL
The *cmd* argument is F_SETOWN and the value of the argument is not valid as a process or process group identifier.

EPERM

The *cmd* argument is F_SETOWN and the calling process does not have permission to send a SIGURG signal to any process specified by *arg*.

The following sections are informative.

EXAMPLES

Locking and Unlocking a File

The following example demonstrates how to place a lock on bytes 100 to 109 of a file and then later remove it. F_SETLK is used to perform a non-blocking lock request so that the process does not have to wait if an incompatible lock is held by another process; instead the process can take some other action.

```
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>

int
main(int argc, char *argv[])
{
    int fd;
    struct flock fl;

    fd = open("testfile", O_RDWR);
    if (fd == -1)
        /* Handle error */;

    /* Make a non-blocking request to place a write lock
       on bytes 100-109 of testfile */

    fl.l_type = F_WRLCK;
    fl.l_whence = SEEK_SET;
    fl.l_start = 100;
    fl.l_len = 10;

    if (fcntl(fd, F_SETLK, &fl) == -1) {
        if (errno == EACCES || errno == EAGAIN) {
            printf("Already locked by another process\n");

            /* We cannot get the lock at the moment */

        } else {
            /* Handle unexpected error */;
        }
    } else { /* Lock was granted... */

        /* Perform I/O on bytes 100 to 109 of file */

        /* Unlock the locked bytes */

        fl.l_type = F_UNLCK;
        fl.l_whence = SEEK_SET;
        fl.l_start = 100;
        fl.l_len = 10;
        if (fcntl(fd, F_SETLK, &fl) == -1)
            /* Handle error */;
    }
    exit(EXIT_SUCCESS);
} /* main */
```

Setting the Close-on-Exec Flag

The following example demonstrates how to set the close-on-exec flag for the file descriptor *fd*.

```
#include <unistd.h>
#include <fcntl.h>
...
    int flags;

    flags = fcntl(fd, F_GETFD);
    if (flags == -1)
        /* Handle error */;
    flags |= FD_CLOEXEC;
    if (fcntl(fd, F_SETFD, flags) == -1)
        /* Handle error */;"
```

APPLICATION USAGE

The *arg* values to F_GETFD, F_SETFD, F_GETFL, and F_SETFL all represent flag values to allow for future growth. Applications using these functions should do a read-modify-write operation on them, rather than assuming that only the values defined by this volume of POSIX.1-2017 are valid. It is a common error to forget this, particularly in the case of F_SETFD. Some implementations set additional file status flags to advise the application of default behavior, even though the application did not request these flags.

On systems which do not perform permission checks at the time of an *fcntl*() call with F_SETOWN, if the permission checks performed at the time the signal is sent disallow sending the signal to any process, the process that called *fcntl*() has no way of discovering that this has happened. A call to *kill*() with signal 0 can be used as a prior check of permissions, although this is no guarantee that permission will be granted at the time a signal is sent, since the target process(es) could change user IDs or privileges in the meantime.

RATIONALE

The ellipsis in the SYNOPSIS is the syntax specified by the ISO C standard for a variable number of arguments. It is used because System V uses pointers for the implementation of file locking functions.

This volume of POSIX.1-2017 permits concurrent read and write access to file data using the *fcntl*() function; this is a change from the 1984 /usr/group standard and early proposals. Without concurrency controls, this feature may not be fully utilized without occasional loss of data.

Data losses occur in several ways. One case occurs when several processes try to update the same record, without sequencing controls; several updates may occur in parallel and the last writer ‘‘wins’’. Another case is a bit-tree or other internal list-based database that is undergoing reorganization. Without exclusive use to the tree segment by the updating process, other reading processes chance getting lost in the database when the index blocks are split, condensed, inserted, or deleted. While *fcntl*() is useful for many applications, it is not intended to be overly general and does not handle the bit-tree example well.

This facility is only required for regular files because it is not appropriate for many devices such as terminals and network connections.

Since *fcntl*() works with ‘‘any file descriptor associated with that file, however it is obtained’’, the file descriptor may have been inherited through a *fork*() or *exec* operation and thus may affect a file that another process also has open.

The use of the open file description to identify what to lock requires extra calls and presents problems if several processes are sharing an open file description, but there are too many implementations of the existing mechanism for this volume of POSIX.1-2017 to use different specifications.

Another consequence of this model is that closing any file descriptor for a given file (whether or not it is the same open file description that created the lock) causes the locks on that file to be relinquished for that process. Equivalently, any close for any file/process pair relinquishes the locks owned on that file for that process. But note that while an open file description may be shared through *fork*(), locks are not inherited through *fork*(). Yet locks may be inherited through one of the *exec* functions.

The identification of a machine in a network environment is outside the scope of this volume of POSIX.1-2017. Thus, an *l_sysid* member, such as found in System V, is not included in the locking structure.

Changing of lock types can result in a previously locked region being split into smaller regions.

Mandatory locking was a major feature of the 1984 /usr/group standard.

For advisory file record locking to be effective, all processes that have access to a file must cooperate and use the advisory mechanism before doing I/O on the file. Enforcement-mode record locking is important when it cannot be assumed that all processes are cooperating. For example, if one user uses an editor to update a file at the same time that a second user executes another process that updates the same file and if only one of the two processes is using advisory locking, the processes are not cooperating. Enforcement-mode record locking would protect against accidental collisions.

Secondly, advisory record locking requires a process using locking to bracket each I/O operation with lock (or test) and unlock operations. With enforcement-mode file and record locking, a process can lock the file once and unlock when all I/O operations have been completed. Enforcement-mode record locking provides a base that can be enhanced; for example, with sharable locks. That is, the mechanism could be enhanced to allow a process to lock a file so other processes could read it, but none of them could write it.

Mandatory locks were omitted for several reasons:

1. Mandatory lock setting was done by multiplexing the set-group-ID bit in most implementations; this was confusing, at best.
2. The relationship to file truncation as supported in 4.2 BSD was not well specified.
3. Any publicly readable file could be locked by anyone. Many historical implementations keep the password database in a publicly readable file. A malicious user could thus prohibit logins. Another possibility would be to hold open a long-distance telephone line.
4. Some demand-paged historical implementations offer memory mapped files, and enforcement cannot be done on that type of file.

Since sleeping on a region is interrupted with any signal, *alarm*() may be used to provide a timeout facility in applications requiring it. This is useful in deadlock detection. Since implementation of full deadlock detection is not always feasible, the **[EDEADLK]** error was made optional.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *close*( ), *exec*, *kill*( ), *open*( ), *sigaction*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fdatasync — synchronize the data of a file (**REALTIME**)

SYNOPSIS

#include <unistd.h>

int fdatasync(int *fildes*);

DESCRIPTION

The *fdatasync*() function shall force all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O completion state.

The functionality shall be equivalent to *fsync*() with the symbol _POSIX_SYNCHRONIZED_IO defined, with the exception that all I/O operations shall be completed as defined for synchronized I/O data integrity completion.

RETURN VALUE

If successful, the *fdatasync*() function shall return the value 0; otherwise, the function shall return the value −1 and set *errno* to indicate the error. If the *fdatasync*() function fails, outstanding I/O operations are not guaranteed to have been completed.

ERRORS

The *fdatasync*() function shall fail if:

EBADF

The *fildes* argument is not a valid file descriptor.

EINVAL

This implementation does not support synchronized I/O for this file.

In the event that any of the queued I/O operations fail, *fdatasync*() shall return the error conditions defined for *read*() and *write*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Note that even if the file descriptor is not open for writing, if there are any pending write requests on the underlying file, then that I/O will be completed prior to the return of *fdatasync*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_fsync( ), *fcntl*( ), *fsync*( ), *open*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fdetach — detach a name from a STREAMS-based file descriptor (**STREAMS**)

SYNOPSIS

#include <stropts.h>

int fdetach(const char **path*);

DESCRIPTION

The *fdetach*() function shall detach a STREAMS-based file from the file to which it was attached by a previous call to *fattach*(). The *path* argument points to the pathname of the attached STREAMS file. The process shall have appropriate privileges or be the owner of the file. A successful call to *fdetach*() shall cause all pathnames that named the attached STREAMS file to again name the file to which the STREAMS file was attached. All subsequent operations on *path* shall operate on the underlying file and not on the STREAMS file.

All open file descriptions established while the STREAMS file was attached to the file referenced by *path* shall still refer to the STREAMS file after the *fdetach*() has taken effect.

If there are no open file descriptors or other references to the STREAMS file, then a successful call to *fdetach*() shall be equivalent to performing the last *close*() on the attached file.

RETURN VALUE

Upon successful completion, *fdetach*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *fdetach*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix.

EINVAL
The *path* argument names a file that is not currently attached.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The effective user ID is not the owner of *path* and the process does not have appropriate privileges.

The *fdetach*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Detaching a File

The following example detaches the STREAMS-based file **/tmp/named-STREAM** from the file to which it was attached by a previous, successful call to *fattach*(). Subsequent calls to open this file refer to the underlying file, not to the STREAMS file.

```
#include <stropts.h>
...
    char *pathname = "/tmp/named-STREAM";
    int ret;

    ret = fdetach(pathname);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *fdetach*() function may be removed in a future version.

SEE ALSO

fattach( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fdim, fdimf, fdiml — compute positive difference between two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fdim(double x, double y);
float fdimf(float x, float y);
long double fdiml(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall determine the positive difference between their arguments. If *x* is greater than *y*, $x-y$ is returned. If *x* is less than or equal to *y*, +0 is returned.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the positive difference value.

If $x-y$ is positive and overflows, a range error shall occur and *fdim*(), *fdimf*(), and *fdiml*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If the correct value would cause underflow, a range error may occur, and *fdim*(), *fdimf*(), and *fdiml*() shall return the correct value, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* or *y* is NaN, a NaN shall be returned.

ERRORS

The *fdim*() function shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The *fdim*() function may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *fmax*( ), *fmin*( )

Section 4.20, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fdopen — associate a stream with a file descriptor

SYNOPSIS

```
#include <stdio.h>

FILE *fdopen(int fildes, const char *mode);
```

DESCRIPTION

The *fdopen*() function shall associate a stream with a file descriptor.

The *mode* argument is a character string having one of the following values:

r or *rb*	Open a file for reading.
w or *wb*	Open a file for writing.
a or *ab*	Open a file for writing at end-of-file.
r+ or *rb*+ or *r+b*	Open a file for update (reading and writing).
w+ or *wb*+ or *w+b*	Open a file for update (reading and writing).
a+ or *ab*+ or *a+b*	Open a file for update (reading and writing) at end-of-file.

The meaning of these flags is exactly as specified in *fopen*(), except that modes beginning with *w* shall not cause truncation of the file.

Additional values for the *mode* argument may be supported by an implementation.

The application shall ensure that the mode of the stream as expressed by the *mode* argument is allowed by the file access mode of the open file description to which *fildes* refers. The file position indicator associated with the new stream is set to the position indicated by the file offset associated with the file descriptor.

The error and end-of-file indicators for the stream shall be cleared. The *fdopen*() function may cause the last data access timestamp of the underlying file to be marked for update.

If *fildes* refers to a shared memory object, the result of the *fdopen*() function is unspecified.

If *fildes* refers to a typed memory object, the result of the *fdopen*() function is unspecified.

The *fdopen*() function shall preserve the offset maximum previously set for the open file description corresponding to *fildes*.

RETURN VALUE

Upon successful completion, *fdopen*() shall return a pointer to a stream; otherwise, a null pointer shall be returned and *errno* set to indicate the error.

ERRORS

The *fdopen*() function shall fail if:

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

The *fdopen*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINVAL
The *mode* argument is not a valid mode.

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

ENOMEM
Insufficient space to allocate a buffer.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

File descriptors are obtained from calls like *open*(), *dup*(), *creat*(), or *pipe*(), which open files but do not return streams.

RATIONALE

The file descriptor may have been obtained from *open*(), *creat*(), *pipe*(), *dup*(), *fcntl*(), or *socket*(); inherited through *fork*(), *posix_spawn*(), or *exec*; or perhaps obtained by other means.

The meanings of the *mode* arguments of *fdopen*() and *fopen*() differ. With *fdopen*(), open for write (*w* or *w*+) does not truncate, and append (*a* or *a*+) cannot create for writing. The *mode* argument formats that include a *b* are allowed for consistency with the ISO C standard function *fopen*(). The *b* has no effect on the resulting stream. Although not explicitly required by this volume of POSIX.1-2017, a good implementation of append (*a*) mode would cause the O_APPEND flag to be set.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.1, *Interaction of File Descriptors and Standard I/O Streams*, *fclose*( ), *fmemopen*( ), *fopen*( ), *open*( ), *open_memstream*( ), *posix_spawn*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fdopendir, opendir — open directory associated with file descriptor

SYNOPSIS

```
#include <dirent.h>

DIR *fdopendir(int fd);
DIR *opendir(const char *dirname);
```

DESCRIPTION

The *fdopendir*() function shall be equivalent to the *opendir*() function except that the directory is specified by a file descriptor rather than by a name. The file offset associated with the file descriptor at the time of the call determines which entries are returned.

Upon successful return from *fdopendir*(), the file descriptor is under the control of the system, and if any attempt is made to close the file descriptor, or to modify the state of the associated description, other than by means of *closedir*(), *readdir*(), *readdir_r*(), *rewinddir*(), or *seekdir*(), the behavior is undefined. Upon calling *closedir*() the file descriptor shall be closed.

It is unspecified whether the FD_CLOEXEC flag will be set on the file descriptor by a successful call to *fdopendir*().

The *opendir*() function shall open a directory stream corresponding to the directory named by the *dirname* argument. The directory stream is positioned at the first entry. If the type **DIR** is implemented using a file descriptor, applications shall only be able to open up to a total of {OPEN_MAX} files and directories.

If the type **DIR** is implemented using a file descriptor, the descriptor shall be obtained as if the O_DIRECTORY flag was passed to *open*().

RETURN VALUE

Upon successful completion, these functions shall return a pointer to an object of type **DIR**. Otherwise, these functions shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *fdopendir*() function shall fail if:

EBADF
The *fd* argument is not a valid file descriptor open for reading.

ENOTDIR
The descriptor *fd* is not associated with a directory.

The *opendir*() function shall fail if:

EACCES
Search permission is denied for the component of the path prefix of *dirname* or read permission is denied for *dirname*.

ELOOP
A loop exists in symbolic links encountered during resolution of the *dirname* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *dirname* does not name an existing directory or *dirname* is an empty string.

ENOTDIR
A component of *dirname* names an existing file that is neither a directory nor a symbolic link to a directory.

The *opendir*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *dirname* argument.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE
Too many files are currently open in the system.

The following sections are informative.

EXAMPLES

Open a Directory Stream

The following program fragment demonstrates how the *opendir*() function is used.

```
#include <dirent.h>
...
    DIR *dir;
    struct dirent *dp;
...
    if ((dir = opendir (".")) == NULL) {
        perror ("Cannot open .");
        exit (1);
    }

    while ((dp = readdir (dir)) != NULL) {
...
```

Find And Open a File

The following program searches through a given directory looking for files whose name does not begin with a dot and whose size is larger than 1 MiB.

```
#include <stdio.h>
#include <dirent.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[])
{
    struct stat statbuf;
    DIR *d;
    struct dirent *dp;
    int dfd, ffd;

    if ((d = fdopendir((dfd = open("./tmp", O_RDONLY)))) == NULL) {
        fprintf(stderr, "Cannot open ./tmp directory\n");
        exit(1);
    }
    while ((dp = readdir(d)) != NULL) {
        if (dp->d_name[0] == '.')
            continue;
        /* there is a possible race condition here as the file
         * could be renamed between the readdir and the open */
        if ((ffd = openat(dfd, dp->d_name, O_RDONLY)) == -1) {
            perror(dp->d_name);
            continue;
        }
        if (fstat(ffd, &statbuf) == 0 && statbuf.st_size > (1024*1024)) {
            /* found it ... */
            printf("%s: %jdK\n", dp->d_name,
                (intmax_t)(statbuf.st_size / 1024));
        }
        close(ffd);
    }
    closedir(d); // note this implicitly closes dfd
    return 0;
}
```

APPLICATION USAGE

The *opendir*() function should be used in conjunction with *readdir*(), *closedir*(), and *rewinddir*() to examine the contents of the directory (see the EXAMPLES section in *readdir*( )). This method is recommended for portability.

RATIONALE

The purpose of the *fdopendir*() function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *opendir*(), resulting in unspecified behavior.

Based on historical implementations, the rules about file descriptors apply to directory streams as well. However, this volume of POSIX.1-2017 does not mandate that the directory stream be implemented using file descriptors. The description of *closedir*() clarifies that if a file descriptor is used for the directory stream, it is mandatory that *closedir*() deallocate the file descriptor. When a file descriptor is used to implement the directory stream, it behaves as if the FD_CLOEXEC had been set for the file descriptor.

The directory entries for dot and dot-dot are optional. This volume of POSIX.1-2017 does not provide a way to test *a priori* for their existence because an application that is portable must be written to look for (and usually ignore) those entries. Writing code that presumes that they are the first two entries does not always work, as many implementations permit them to be other than the first two entries, with a "normal" entry preceding them. There is negligible value in providing a way to determine what the implementation does because the code to deal with dot and dot-dot must be written in any case and because such a flag would add to the list of those flags (which has proven in itself to be objectionable) and might be abused.

Since the structure and buffer allocation, if any, for directory operations are defined by the implementation, this volume of POSIX.1-2017 imposes no portability requirements for erroneous program constructs, erroneous data, or the use of unspecified values such as the use or referencing of a *dirp* value or a **dirent** structure value after a directory stream has been closed or after a *fork*() or one of the *exec* function calls.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *dirfd*( ), *fstatat*( ), *open*( ), *readdir*( ), *rewinddir*( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

feclearexcept — clear floating-point exception

SYNOPSIS

```
#include <fenv.h>

int feclearexcept(int excepts);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *feclearexcept*() function shall attempt to clear the supported floating-point exceptions represented by *excepts*.

RETURN VALUE

If the argument is zero or if all the specified exceptions were successfully cleared, *feclearexcept*() shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetexceptflag( ), *feraiseexcept*( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fegetenv, fesetenv — get and set current floating-point environment

SYNOPSIS

```
#include <fenv.h>

int fegetenv(fenv_t *envp);
int fesetenv(const fenv_t *envp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fegetenv*() function shall attempt to store the current floating-point environment in the object pointed to by *envp*.

The *fesetenv*() function shall attempt to establish the floating-point environment represented by the object pointed to by *envp*. The argument *envp* shall point to an object set by a call to *fegetenv*() or *feholdexcept*(), or equal a floating-point environment macro. The *fesetenv*() function does not raise floating-point exceptions, but only installs the state of the floating-point status flags represented through its argument.

RETURN VALUE

If the representation was successfully stored, *fegetenv*() shall return zero. Otherwise, it shall return a non-zero value. If the environment was successfully established, *fesetenv*() shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feholdexcept( ), *feupdateenv*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fegetexceptflag, fesetexceptflag — get and set floating-point status flags

SYNOPSIS

#include <fenv.h>

int fegetexceptflag(fexcept_t **flagp*, int *excepts*);
int fesetexceptflag(const fexcept_t **flagp*, int *excepts*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fegetexceptflag*() function shall attempt to store an implementation-defined representation of the states of the floating-point status flags indicated by the argument *excepts* in the object pointed to by the argument *flagp*.

The *fesetexceptflag*() function shall attempt to set the floating-point status flags indicated by the argument *excepts* to the states stored in the object pointed to by *flagp*. The value pointed to by *flagp* shall have been set by a previous call to *fegetexceptflag*() whose second argument represented at least those floating-point exceptions represented by the argument *excepts*. This function does not raise floating-point exceptions, but only sets the state of the flags.

RETURN VALUE

If the representation was successfully stored, *fegetexceptflag*() shall return zero. Otherwise, it shall return a non-zero value. If the *excepts* argument is zero or if all the specified exceptions were successfully set, *fesetexceptflag*() shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *feraiseexcept*( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fegetround, fesetround — get and set current rounding direction

SYNOPSIS

```
#include <fenv.h>

int fegetround(void);
int fesetround(int round);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fegetround*() function shall get the current rounding direction.

The *fesetround*() function shall establish the rounding direction represented by its argument *round*. If the argument is not equal to the value of a rounding direction macro, the rounding direction is not changed.

RETURN VALUE

The *fegetround*() function shall return the value of the rounding direction macro representing the current rounding direction or a negative value if there is no such rounding direction macro or the current rounding direction is not determinable.

The *fesetround*() function shall return a zero value if and only if the requested rounding direction was established.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example saves, sets, and restores the rounding direction, reporting an error and aborting if setting the rounding direction fails:

```
#include <fenv.h>
#include <assert.h>
void f(int round_dir)
{
   #pragma STDC FENV_ACCESS ON
   int save_round;
   int setround_ok;
   save_round = fegetround();
   setround_ok = fesetround(round_dir);
   assert(setround_ok == 0);
   /* ... */
   fesetround(save_round);
   /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

feholdexcept — save current floating-point environment

SYNOPSIS

```
#include <fenv.h>

int feholdexcept(fenv_t *envp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *feholdexcept*() function shall save the current floating-point environment in the object pointed to by *envp*, clear the floating-point status flags, and then install a non-stop (continue on floating-point exceptions) mode, if available, for all floating-point exceptions.

RETURN VALUE

The *feholdexcept*() function shall return zero if and only if non-stop floating-point exception handling was successfully installed.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *feholdexcept*() function should be effective on typical IEC 60559: 1989 standard implementations which have the default non-stop mode and at least one other mode for trap handling or aborting. If the implementation provides only the non-stop mode, then installing the non-stop mode is trivial.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetenv( ), *feupdateenv*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

feof — test end-of-file indicator on a stream

SYNOPSIS

```
#include <stdio.h>

int feof(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *feof*() function shall test the end-of-file indicator for the stream pointed to by *stream*.

The *feof*() function shall not change the setting of *errno* if *stream* is valid.

RETURN VALUE

The *feof*() function shall return non-zero if and only if the end-of-file indicator is set for *stream*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clearerr( ), *ferror*( ), *fopen*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

feraiseexcept — raise floating-point exception

SYNOPSIS

#include <fenv.h>

int feraiseexcept(int *excepts*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *feraiseexcept*() function shall attempt to raise the supported floating-point exceptions represented by the *excepts* argument. The order in which these floating-point exceptions are raised is unspecified, except that if the *excepts* argument represents IEC 60559 valid coincident floating-point exceptions for atomic operations (namely overflow and inexact, or underflow and inexact), then overflow or underflow shall be raised before inexact. Whether the *feraiseexcept*() function additionally raises the inexact floating-point exception whenever it raises the overflow or underflow floating-point exception is implementation-defined.

RETURN VALUE

If the argument is zero or if all the specified exceptions were successfully raised, *feraiseexcept*() shall return zero. Otherwise, it shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The effect is intended to be similar to that of floating-point exceptions raised by arithmetic operations. Hence, enabled traps for floating-point exceptions raised by this function are taken.

RATIONALE

Raising overflow or underflow is allowed to also raise inexact because on some architectures the only practical way to raise an exception is to execute an instruction that has the exception as a side-effect. The function is not restricted to accept only valid coincident expressions for atomic operations, so the function can be used to raise exceptions accrued over several operations.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fegetexceptflag*( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ferror — test error indicator on a stream

SYNOPSIS

```
#include <stdio.h>

int ferror(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ferror*() function shall test the error indicator for the stream pointed to by *stream*.

The *ferror*() function shall not change the setting of *errno* if *stream* is valid.

RETURN VALUE

The *ferror*() function shall return non-zero if and only if the error indicator is set for *stream*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

clearerr( ), *feof*( ), *fopen*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fesetenv — set current floating-point environment

SYNOPSIS

```
#include <fenv.h>

int fesetenv(const fenv_t *envp);
```

DESCRIPTION

Refer to *fegetenv*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fesetexceptflag — set floating-point status flags

SYNOPSIS

#include <fenv.h>

int fesetexceptflag(const fexcept_t **flagp*, int *excepts*);

DESCRIPTION

Refer to *fegetexceptflag*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fesetround — set current rounding direction

SYNOPSIS

#include <fenv.h>

int fesetround(int *round*);

DESCRIPTION

Refer to *fegetround*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fetestexcept — test floating-point exception flags

SYNOPSIS

```
#include <fenv.h>

int fetestexcept(int excepts);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fetestexcept*() function shall determine which of a specified subset of the floating-point exception flags are currently set. The *excepts* argument specifies the floating-point status flags to be queried.

RETURN VALUE

The *fetestexcept*() function shall return the value of the bitwise-inclusive OR of the floating-point exception macros corresponding to the currently set floating-point exceptions included in *excepts*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example calls function *f*() if an invalid exception is set, and then function *g*() if an overflow exception is set:

```
#include <fenv.h>
/* ... */
{
    #pragma STDC FENV_ACCESS ON
    int set_excepts;
    feclearexcept(FE_INVALID | FE_OVERFLOW);
    // maybe raise exceptions
    set_excepts = fetestexcept(FE_INVALID | FE_OVERFLOW);
    if (set_excepts & FE_INVALID) f();
    if (set_excepts & FE_OVERFLOW) g();
    /* ... */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fegetexceptflag*( ), *feraiseexcept*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

feupdateenv — update floating-point environment

SYNOPSIS

```
#include <fenv.h>

int feupdateenv(const fenv_t *envp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *feupdateenv*() function shall attempt to save the currently raised floating-point exceptions in its automatic storage, attempt to install the floating-point environment represented by the object pointed to by *envp*, and then attempt to raise the saved floating-point exceptions. The argument *envp* shall point to an object set by a call to *feholdexcept*() or *fegetenv*(), or equal a floating-point environment macro.

RETURN VALUE

The *feupdateenv*() function shall return a zero value if and only if all the required actions were successfully carried out.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following example shows sample code to hide spurious underflow floating-point exceptions:

```
#include <fenv.h>
double f(double x)
{
    #pragma STDC FENV_ACCESS ON
    double result;
    fenv_t save_env;
    feholdexcept(&save_env);
    // compute result
    if (/* test spurious underflow */)
    feclearexcept(FE_UNDERFLOW);
    feupdateenv(&save_env);
    return result;
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fegetenv( ), *feholdexcept*( )

The Base Definitions volume of POSIX.1-2017, **<fenv.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fexecve — execute a file

SYNOPSIS

#include <unistd.h>

int fexecve(int *fd*, char *const *argv[]*, char *const *envp[]*);

DESCRIPTION

Refer to *exec*.

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fflush — flush a stream

SYNOPSIS

```
#include <stdio.h>

int fflush(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *stream* points to an output stream or an update stream in which the most recent operation was not input, *fflush*() shall cause any unwritten data for that stream to be written to the file, and the last data modification and last file status change timestamps of the underlying file shall be marked for update.

For a stream open for reading with an underlying file description, if the file is not already at EOF, and the file is one capable of seeking, the file offset of the underlying open file description shall be set to the file position of the stream, and any characters pushed back onto the stream by *ungetc*() or *ungetwc*() that have not subsequently been read from the stream shall be discarded (without further changing the file offset).

If *stream* is a null pointer, *fflush*() shall perform this flushing action on all streams for which the behavior is defined above.

RETURN VALUE

Upon successful completion, *fflush*() shall return 0; otherwise, it shall set the error indicator for the stream, return EOF, and set *errno* to indicate the error.

ERRORS

The *fflush*() function shall fail if:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying *stream* is not valid.

EFBIG
An attempt was made to write a file that exceeds the maximum file size.

EFBIG
An attempt was made to write a file that exceeds the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR
The *fflush*() function was interrupted by a signal.

EIO The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOMEM
The underlying stream was created by *open_memstream*() or *open_wmemstream*() and insufficient memory is available.

ENOSPC
There was no free space remaining on the device containing the file or in the buffer used by the *fmemopen*() function.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fflush*() function may fail if:

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

Sending Prompts to Standard Output

The following example uses *printf*() calls to print a series of prompts for information the user must enter from standard input. The *fflush*() calls force the output to standard output. The *fflush*() function is used because standard output is usually buffered and the prompt may not immediately be printed on the output or terminal. The *getline*() function calls read strings from standard input and place the results in variables, for use later in the program.

```
char *user;
char *oldpasswd;
char *newpasswd;
ssize_t llen;
size_t blen;
struct termios term;
tcflag_t saveflag;

printf("User name: ");
fflush(stdout);
blen = 0;
llen = getline(&user, &blen, stdin);
user[llen-1] = 0;
tcgetattr(fileno(stdin), &term);
saveflag = term.c_lflag;
term.c_lflag &= ~ECHO;
tcsetattr(fileno(stdin), TCSANOW, &term);
printf("Old password: ");
fflush(stdout);
blen = 0;
llen = getline(&oldpasswd, &blen, stdin);
oldpasswd[llen-1] = 0;

printf("\nNew password: ");
fflush(stdout);
blen = 0;
llen = getline(&newpasswd, &blen, stdin);
newpasswd[llen-1] = 0;
term.c_lflag = saveflag;
tcsetattr(fileno(stdin), TCSANOW, &term);
free(user);
free(oldpasswd);
free(newpasswd);
```

APPLICATION USAGE

None.

RATIONALE

Data buffered by the system may make determining the validity of the position of the current file descriptor impractical. Thus, enforcing the repositioning of the file descriptor after *fflush*() on streams open for *read*() is not mandated by POSIX.1-2008.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fmemopen*( ), *getrlimit*( ), *open_memstream*( ), *ulimit*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ffs — find first set bit

SYNOPSIS

```
#include <strings.h>

int ffs(int i);
```

DESCRIPTION

The *ffs*() function shall find the first bit set (beginning with the least significant bit) in *i*, and return the index of that bit. Bits are numbered starting at one (the least significant bit).

RETURN VALUE

The *ffs*() function shall return the index of the first bit set. If *i* is 0, then *ffs*() shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<strings.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fgetc — get a byte from a stream

SYNOPSIS

#include <stdio.h>

int fgetc(FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If the end-of-file indicator for the input stream pointed to by *stream* is not set and a next byte is present, the *fgetc*() function shall obtain the next byte as an **unsigned char** converted to an **int**, from the input stream pointed to by *stream*, and advance the associated file position indicator for the stream (if defined). Since *fgetc*() operates on bytes, reading a character consisting of multiple bytes (or ''a multi-byte character'') may require multiple calls to *fgetc*().

The *fgetc*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *fscanf*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, *fgetc*() shall return the next byte from the input stream pointed to by *stream*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetc*() shall return EOF. If a read error occurs, the error indicator for the stream shall be set, *fgetc*() shall return EOF, and shall set *errno* to indicate the error.

ERRORS

The *fgetc*() function shall fail if data needs to be read and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the *fgetc*() operation.

EBADF
The file descriptor underlying *stream* is not a valid file descriptor open for reading.

EINTR
The read operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

EOVERFLOW
The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The *fgetc*() function may fail if:

ENOMEM
Insufficient storage space is available.

ENXIO

A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *fgetc*() is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

The *ferror*() or *feof*() functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *feof*( ), *ferror*( ), *fgets*( ), *fread*( ), *fscanf*( ), *getchar*( ), *getc*( ), *gets*( ), *ungetc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fgetpos — get current file position information

SYNOPSIS

```
#include <stdio.h>

int fgetpos(FILE *restrict stream, fpos_t *restrict pos);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fgetpos*() function shall store the current values of the parse state (if any) and file position indicator for the stream pointed to by *stream* in the object pointed to by *pos*. The value stored contains unspecified information usable by *fsetpos*() for repositioning the stream to its position at the time of the call to *fgetpos*().

The *fgetpos*() function shall not change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, *fgetpos*() shall return 0; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *fgetpos*() function shall fail if:

EBADF
The file descriptor underlying *stream* is not valid.

EOVERFLOW
The current value of the file position cannot be represented correctly in an object of type **fpos_t**.

ESPIPE
The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *ftell*( ), *rewind*( ), *ungetc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fgets — get a string from a stream

SYNOPSIS

#include <stdio.h>

char *fgets(char *restrict *s*, int *n*, FILE *restrict *stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fgets*() function shall read bytes from *stream* into the array pointed to by *s* until *n*−1 bytes are read, or a <newline> is read and transferred to *s*, or an end-of-file condition is encountered. A null byte shall be written immediately after the last byte read into the array. If the end-of-file condition is encountered before any bytes are read, the contents of the array pointed to by *s* shall not be changed.

The *fgets*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *fscanf*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, *fgets*() shall return *s*. If the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgets*() shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set, *fgets*() shall return a null pointer, and shall set *errno* to indicate the error.

ERRORS

Refer to *fgetc*( ).

The following sections are informative.

EXAMPLES

Reading Input

The following example uses *fgets*() to read lines of input. It assumes that the file it is reading is a text file and that lines in this text file are no longer than 16384 (or {LINE_MAX} if it is less than 16384 on the implementation where it is running) bytes long. (Note that the standard utilities have no line length limit if *sysconf*(_SC_LINE_MAX) returns −1 without setting *errno*. This example assumes that *sysconf*(_SC_LINE_MAX) will not fail.)

```
#include <limits.h>
#include <stdio.h>
#include <unistd.h>
#define MYLIMIT 16384

char *line;
int line_max;
if (LINE_MAX >= MYLIMIT) {
    // Use maximum line size of MYLIMIT. If LINE_MAX is
    // bigger than our limit, sysconf() cannot report a
    // smaller limit.
    line_max = MYLIMIT;
} else {
    long limit = sysconf(_SC_LINE_MAX);
    line_max = (limit < 0 || limit > MYLIMIT) ? MYLIMIT : (int)limit;
}

// line_max + 1 leaves room for the null byte added by fgets().
line = malloc(line_max + 1);
if (line == NULL) {
    // out of space
    ...
    return error;
}

while (fgets(line, line_max + 1, fp) != NULL) {
    // Verify that a full line has been read ...
    // If not, report an error or prepare to treat the
    // next time through the loop as a read of a
    // continuation of the current line.
    ...
    // Process line ...
    ...
}
free(line);
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetc*( ), *fopen*( ), *fread*( ), *fscanf*( ), *getc*( ), *getchar*( ), *getdelim*( ), *gets*( ), *ungetc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fgetwc — get a wide-character code from a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t fgetwc(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fgetwc*() function shall obtain the next character (if present) from the input stream pointed to by *stream*, convert that to the corresponding wide-character code, and advance the associated file position indicator for the stream (if defined).

If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

The *fgetwc*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*(), *fgetws*(), *fwscanf*(), *getwc*(), *getwchar*(), *vfwscanf*(), *vwscanf*(), or *wscanf*() using *stream* that returns data not supplied by a prior call to *ungetwc*().

The *fgetwc*() function shall not change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, the *fgetwc*() function shall return the wide-character code of the character read from the input stream pointed to by *stream* converted to a type **wint_t**. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetwc*() shall return WEOF. If a read error occurs, the error indicator for the stream shall be set, *fgetwc*() shall return WEOF, and shall set *errno* to indicate the error. If an encoding error occurs, the error indicator for the stream shall be set, *fgetwc*() shall return WEOF, and shall set *errno* to indicate the error.

ERRORS

The *fgetwc*() function shall fail if data needs to be read and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the *fgetwc*() operation.

EBADF
The file descriptor underlying *stream* is not a valid file descriptor open for reading.

EILSEQ
The data obtained from the input stream does not form a valid character.

EINTR
The read operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is in a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

EOVERFLOW
The file is a regular file and an attempt was made to read at or beyond the offset maximum associated with the corresponding stream.

The *fgetwc*() function may fail if:

ENOMEM
Insufficient storage space is available.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *ferror*() or *feof*() functions must be used to distinguish between an error condition and an end-of-file condition.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *feof*( ), *ferror*( ), *fopen*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fgetws — get a wide-character string from a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wchar_t *fgetws(wchar_t *restrict ws, int n,
   FILE *restrict stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fgetws*() function shall read characters from the *stream*, convert these to the corresponding wide-character codes, place them in the **wchar_t** array pointed to by *ws*, until *n*−1 characters are read, or a <newline> is read, converted, and transferred to *ws*, or an end-of-file condition is encountered. The wide-character string, *ws*, shall then be terminated with a null wide-character code.

If an error occurs, the resulting value of the file position indicator for the stream is unspecified.

The *fgetws*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*(), *fgetws*(), *fwscanf*(), *getwc*(), *getwchar*(), *vfwscanf*(), *vwscanf*(), or *wscanf*() using *stream* that returns data not supplied by a prior call to *ungetwc*().

RETURN VALUE

Upon successful completion, *fgetws*() shall return *ws*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *fgetws*() shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set, *fgetws*() shall return a null pointer, and shall set *errno* to indicate the error.

ERRORS

Refer to *fgetwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *fread*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fileno — map a stream pointer to a file descriptor

SYNOPSIS

```
#include <stdio.h>

int fileno(FILE *stream);
```

DESCRIPTION

The *fileno*() function shall return the integer file descriptor associated with the stream pointed to by *stream*.

RETURN VALUE

Upon successful completion, *fileno*() shall return the integer value of the file descriptor associated with *stream*. Otherwise, the value −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fileno*() function shall fail if:

EBADF
The stream is not associated with a file.

The *fileno*() function may fail if:

EBADF
The file descriptor underlying *stream* is not a valid file descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Without some specification of which file descriptors are associated with these streams, it is impossible for an application to set up the streams for another application it starts with *fork*() and *exec*. In particular, it would not be possible to write a portable version of the *sh* command interpreter (although there may be other constraints that would prevent that portability).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5.1, *Interaction of File Descriptors and Standard I/O Streams*, *dirfd*( ), *fdopen*( ), *fopen*( ), *stdin*

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

flockfile, ftrylockfile, funlockfile — stdio locking functions

SYNOPSIS

#include <stdio.h>

void flockfile(FILE **file*);
int ftrylockfile(FILE **file*);
void funlockfile(FILE **file*);

DESCRIPTION

These functions shall provide for explicit application-level locking of stdio (**FILE ***) objects. These functions can be used by a thread to delineate a sequence of I/O statements that are executed as a unit.

The *flockfile*() function shall acquire for a thread ownership of a (**FILE ***) object.

The *ftrylockfile*() function shall acquire for a thread ownership of a (**FILE ***) object if the object is available; *ftrylockfile*() is a non-blocking version of *flockfile*().

The *funlockfile*() function shall relinquish the ownership granted to the thread. The behavior is undefined if a thread other than the current owner calls the *funlockfile*() function.

The functions shall behave as if there is a lock count associated with each (**FILE ***) object. This count is implicitly initialized to zero when the (**FILE ***) object is created. The (**FILE ***) object is unlocked when the count is zero. When the count is positive, a single thread owns the (**FILE ***) object. When the *flockfile*() function is called, if the count is zero or if the count is positive and the caller owns the (**FILE ***) object, the count shall be incremented. Otherwise, the calling thread shall be suspended, waiting for the count to return to zero. Each call to *funlockfile*() shall decrement the count. This allows matching calls to *flockfile*() (or successful calls to *ftrylockfile*()) and *funlockfile*() to be nested.

All functions that reference (**FILE ***) objects, except those with names ending in *_unlocked*, shall behave as if they use *flockfile*() and *funlockfile*() internally to obtain ownership of these (**FILE ***) objects.

RETURN VALUE

None for *flockfile*() and *funlockfile*().

The *ftrylockfile*() function shall return zero for success and non-zero to indicate that the lock cannot be acquired.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

A call to *exit*() can block until locked streams are unlocked because a thread having ownership of a (**FILE***) object blocks all function calls that reference that (**FILE***) object (except those with names ending in _unlocked) from other threads, including calls to *exit*().

RATIONALE

The *flockfile*() and *funlockfile*() functions provide an orthogonal mutual-exclusion lock for each **FILE**. The *ftrylockfile*() function provides a non-blocking attempt to acquire a file lock, analogous to *pthread_mutex_trylock*().

These locks behave as if they are the same as those used internally by *stdio* for thread-safety. This both provides thread-safety of these functions without requiring a second level of internal locking and allows functions in *stdio* to be implemented in terms of other *stdio* functions.

Application developers and implementors should be aware that there are potential deadlock problems on **FILE** objects. For example, the line-buffered flushing semantics of *stdio* (requested via {_IOLBF}) require that certain input operations sometimes cause the buffered contents of implementation-defined line-buffered output streams to be flushed. If two threads each hold the lock on the other's **FILE**, deadlock ensues. This type of deadlock can be avoided by acquiring **FILE** locks in a consistent order. In particular, the line-buffered output stream deadlock can typically be avoided by acquiring locks on input streams before locks on output streams if a thread would be acquiring both.

In summary, threads sharing *stdio* streams with other threads can use *flockfile*() and *funlockfile*() to cause sequences of I/O performed by a single thread to be kept bundled. The only case where the use of *flockfile*() and *funlockfile*() is required is to provide a scope protecting uses of the **_unlocked* functions/macros. This moves the cost/performance tradeoff to the optimal point.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *getc_unlocked*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

floor, floorf, floorl — floor function

SYNOPSIS

```
#include <math.h>

double floor(double x);
float floorf(float x);
long double floorl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the largest integral value not greater than x.

RETURN VALUE

The result shall have the same sign as x.

Upon successful completion, these functions shall return the largest integral value not greater than x, expressed as a **double**, **float**, or **long double**, as appropriate for the return type of the function.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions might not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

These functions may raise the inexact floating-point exception if the result differs in value from the argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ceil( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

Section 4.20, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fma, fmaf, fmal — floating-point multiply-add

SYNOPSIS

```
#include <math.h>

double fma(double x, double y, double z);
float fmaf(float x, float y, float z);
long double fmal(long double x, long double y, long double z);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute (*x* * *y*) + *z*, rounded as one ternary operation: they shall compute the value (as if) to infinite precision and round once to the result format, according to the rounding mode characterized by the value of FLT_ROUNDS.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return (*x* * *y*) + *z*, rounded as one ternary operation.

If the result overflows or underflows, a range error may occur. On systems that support the IEC 60559 Floating-Point option, if the result overflows a range error shall occur.

If *x* or *y* are NaN, a NaN shall be returned.

If *x* multiplied by *y* is an exact infinity and *z* is also an infinity but with the opposite sign, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of *x* and *y* is infinite, the other is zero, and *z* is not a NaN, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If one of *x* and *y* is infinite, the other is zero, and *z* is a NaN, a NaN shall be returned and a domain error may occur.

If *x***y* is not 0*Inf nor Inf*0 and *z* is a NaN, a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error
: The value of *x***y*+*z* is invalid, or the value *x***y* is invalid and *z* is not a NaN.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Range Error
: The result overflows.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Domain Error
The value $x*y$ is invalid and z is a NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

In many cases, clever use of floating (*fused*) multiply-add leads to much improved code; but its unexpected use by the compiler can undermine carefully written code. The FP_CONTRACT macro can be used to disallow use of floating multiply-add; and the *fma*() function guarantees its use where desired. Many current machines provide hardware floating multiply-add instructions; software implementation can be used for others.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fmax, fmaxf, fmaxl — determine maximum numeric value of two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fmax(double x, double y);
float fmaxf(float x, float y);
long double fmaxl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall determine the maximum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

RETURN VALUE

Upon successful completion, these functions shall return the maximum numeric value of their arguments.

If just one argument is a NaN, the other argument shall be returned.

If *x* and *y* are NaN, a NaN shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdim( ), *fmin*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fmemopen — open a memory buffer stream

SYNOPSIS

```
#include <stdio.h>

FILE *fmemopen(void *restrict buf, size_t size,
   const char *restrict mode);
```

DESCRIPTION

The *fmemopen*() function shall associate the buffer given by the *buf* and *size* arguments with a stream. The *buf* argument shall be either a null pointer or point to a buffer that is at least *size* bytes long.

The *mode* argument points to a string. If the string is one of the following, the stream shall be opened in the indicated mode. Otherwise, the behavior is undefined.

r	Open the stream for reading.
w	Open the stream for writing.
a	Append; open the stream for writing at the first null byte.
r+	Open the stream for update (reading and writing).
w+	Open the stream for update (reading and writing). Truncate the buffer contents.
a+	Append; open the stream for update (reading and writing); the initial position is at the first null byte.

Implementations shall accept all mode strings allowed by *fopen*(), but the use of the character **'b'** shall produce implementation-defined results, where the resulting **FILE *** need not behave the same as if **'b'** were omitted.

If a null pointer is specified as the *buf* argument, *fmemopen*() shall allocate *size* bytes of memory as if by a call to *malloc*(). This buffer shall be automatically freed when the stream is closed. Because this feature is only useful when the stream is opened for updating (because there is no way to get a pointer to the buffer) the *fmemopen*() call may fail if the *mode* argument does not include a **'+'**.

The stream shall maintain a current position in the buffer. This position shall be initially set to either the beginning of the buffer (for *r* and *w* modes) or to the first null byte in the buffer (for *a* modes). If no null byte is found in append mode, the initial position shall be set to one byte after the end of the buffer.

If *buf* is a null pointer, the initial position shall always be set to the beginning of the buffer.

The stream shall also maintain the size of the current buffer contents; use of *fseek*() or *fseeko*() on the stream with SEEK_END shall seek relative to this size. For modes *r* and *r+* the size shall be set to the value given by the *size* argument. For modes *w* and *w+* the initial size shall be zero and for modes *a* and *a+* the initial size shall be:

* Zero, if *buf* is a null pointer
* The position of the first null byte in the buffer, if one is found
* The value of the *size* argument, if *buf* is not a null pointer and no null byte is found

A read operation on the stream shall not advance the current buffer position beyond the current buffer size. Reaching the buffer size in a read operation shall count as ``end-of-file''. Null bytes in the buffer shall have no special meaning for reads. The read operation shall start at the current buffer position of the stream.

A write operation shall start either at the current position of the stream (if *mode* has not specified **'a'** as the first character) or at the current size of the stream (if *mode* had **'a'** as the first character). If the current position at the end of the write is larger than the current buffer size, the current buffer size shall be set to the current position. A write operation on the stream shall not advance the current buffer size beyond the size given in the *size* argument.

When a stream open for writing is flushed or closed, a null byte shall be written at the current position or at the end of the buffer, depending on the size of the contents. If a stream open for update is flushed or closed and the last write has advanced the current buffer size, a null byte shall be written at the end of the buffer if it fits.

An attempt to seek a memory buffer stream to a negative position or to a position larger than the buffer size given in the *size* argument shall fail.

RETURN VALUE

Upon successful completion, *fmemopen*() shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

The *fmemopen*() function shall fail if:

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

The *fmemopen*() function may fail if:

EINVAL
The value of the *mode* argument is not valid.

EINVAL
The *buf* argument is a null pointer and the *mode* argument does not include a **'+'** character.

EINVAL
The *size* argument specifies a buffer size of zero and the implementation does not support this.

ENOMEM
The *buf* argument is a null pointer and the allocation of a buffer of length *size* has failed.

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

The following sections are informative.

EXAMPLES

```
#include <stdio.h>
#include <string.h>

static char buffer[] = "foobar";

int
main (void)
{
  int ch;
  FILE *stream;

  stream = fmemopen(buffer, strlen (buffer), "r");
  if (stream == NULL)
    /* handle error */;

  while ((ch = fgetc(stream)) != EOF)
    printf("Got %c\n", ch);

  fclose(stream);
  return (0);
}
```

This program produces the following output:

```
Got f
Got o
Got o
Got b
Got a
Got r
```

APPLICATION USAGE

None.

RATIONALE

This interface has been introduced to eliminate many of the errors encountered in the construction of strings, notably overflowing of strings. This interface prevents overflow.

FUTURE DIRECTIONS

A future version of this standard may mandate specific behavior when the *mode* argument includes **'b'**.

A future version of this standard may require support of zero-length buffer streams explicitly.

SEE ALSO

fdopen( ), *fopen*( ), *freopen*( ), *fseek*( ), *malloc*( ), *open_memstream*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fmin, fminf, fminl — determine minimum numeric value of two floating-point numbers

SYNOPSIS

```
#include <math.h>

double fmin(double x, double y);
float fminf(float x, float y);
long double fminl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall determine the minimum numeric value of their arguments. NaN arguments shall be treated as missing data: if one argument is a NaN and the other numeric, then these functions shall choose the numeric value.

RETURN VALUE

Upon successful completion, these functions shall return the minimum numeric value of their arguments.

If just one argument is a NaN, the other argument shall be returned.

If *x* and *y* are NaN, a NaN shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdim( ), *fmax*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fmod, fmodf, fmodl — floating-point remainder value function

SYNOPSIS

```
#include <math.h>

double fmod(double x, double y);
float fmodf(float x, float y);
long double fmodl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall return the floating-point remainder of the division of *x* by *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return the value *x*−*i***y*, for some integer *i* such that, if *y* is non-zero, the result has the same sign as *x* and magnitude less than the magnitude of *y*.

If the correct value would cause underflow, and is not representable, a range error may occur, and *fmod*(), *modf*(), and *fmodl*() shall return 0.0, or (if the IEC 60559 Floating-Point option is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* or *y* is NaN, a NaN shall be returned, and none of the conditions below shall be considered.

If *y* is zero, a domain error shall occur, and a NaN shall be returned.

If *x* is infinite, a domain error shall occur, and a NaN shall be returned.

If *x* is ±0 and *y* is not zero, ±0 shall be returned.

If *x* is not infinite and *y* is ±Inf, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Domain Error
The *x* argument is infinite or *y* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( )

Section 4.20, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fmtmsg — display a message in the specified format on standard error and/or a system console

SYNOPSIS

#include <fmtmsg.h>

int fmtmsg(long *classification*, const char **label*, int *severity*,
  const char **text*, const char **action*, const char **tag*);

DESCRIPTION

The *fmtmsg*() function shall display messages in a specified format instead of the traditional *printf*() function.

Based on a message's classification component, *fmtmsg*() shall write a formatted message either to standard error, to the console, or to both.

A formatted message consists of up to five components as defined below. The component *classification* is not part of a message displayed to the user, but defines the source of the message and directs the display of the formatted message.

classification Contains the sum of identifying values constructed from the constants defined below. Any one identifier from a subclass may be used in combination with a single identifier from a different subclass. Two or more identifiers from the same subclass should not be used together, with the exception of identifiers from the display subclass. (Both display subclass identifiers may be used so that messages can be displayed to both standard error and the system console.)

Major Classifications
Identifies the source of the condition. Identifiers are: MM_HARD (hardware), MM_SOFT (software), and MM_FIRM (firmware).

Message Source Subclassifications
Identifies the type of software in which the problem is detected. Identifiers are: MM_APPL (application), MM_UTIL (utility), and MM_OPSYS (operating system).

Display Subclassifications
Indicates where the message is to be displayed. Identifiers are: MM_PRINT to display the message on the standard error stream, MM_CONSOLE to display the message on the system console. One or both identifiers may be used.

Status Subclassifications
Indicates whether the application can recover from the condition. Identifiers are: MM_RECOVER (recoverable) and MM_NRECOV (non-recoverable).

An additional identifier, MM_NULLMC, indicates that no classification component is supplied for the message.

label Identifies the source of the message. The format is two fields separated by a <colon>. The first field is up to 10 bytes, the second is up to 14 bytes.

severity Indicates the seriousness of the condition. Identifiers for the levels of *severity* are:

MM_HALT Indicates that the application has encountered a severe fault and is halting. Produces the string **"HALT"**.

MM_ERROR
Indicates that the application has detected a fault. Produces the string **"ERROR"**.

MM_WARNING
: Indicates a condition that is out of the ordinary, that might be a problem, and should be watched. Produces the string **"WARNING"**.

MM_INFO
: Provides information about a condition that is not in error. Produces the string **"INFO"**.

MM_NOSEV
: Indicates that no severity level is supplied for the message.

text
: Describes the error condition that produced the message. The character string is not limited to a specific size. If the character string is empty, then the text produced is unspecified.

action
: Describes the first step to be taken in the error-recovery process. The *fmtmsg*() function precedes the action string with the prefix: **"TO**FIX:"**.** The *action* string is not limited to a specific size.

tag
: An identifier that references on-line documentation for the message. Suggested usage is that *tag* includes the *label* and a unique identifying number. A sample *tag* is **"XSI:cat:146"**.

The *MSGVERB* environment variable (for message verbosity) shall determine for *fmtmsg*() which message components it is to select when writing messages to standard error. The value of *MSGVERB* shall be a <colon>-separated list of optional keywords. Valid keywords are: *label*, *severity*, *text*, *action*, and *tag*. If *MSGVERB* contains a keyword for a component and the component's value is not the component's null value, *fmtmsg*() shall include that component in the message when writing the message to standard error. If *MSGVERB* does not include a keyword for a message component, that component shall not be included in the display of the message. The keywords may appear in any order. If *MSGVERB* is not defined, if its value is the null string, if its value is not of the correct format, or if it contains keywords other than the valid ones listed above, *fmtmsg*() shall select all components.

MSGVERB shall determine which components are selected for display to standard error. All message components shall be included in console messages.

RETURN VALUE

The *fmtmsg*() function shall return one of the following values:

MM_OK
: The function succeeded.

MM_NOTOK
: The function failed completely.

MM_NOMSG
: The function was unable to generate a message on standard error, but otherwise succeeded.

MM_NOCON
: The function was unable to generate a console message, but otherwise succeeded.

ERRORS

None.

The following sections are informative.

EXAMPLES

1. The following example of *fmtmsg*():

```
fmtmsg(MM_PRINT, "XSI:cat", MM_ERROR, "illegal option",
"refer to cat in user's reference manual", "XSI:cat:001")
```

   produces a complete message in the specified message format:

```
XSI:cat: ERROR: illegal option
TO FIX: refer to cat in user's reference manual XSI:cat:001
```

2. When the environment variable *MSGVERB* is set as follows:

   ```
   MSGVERB=severity:text:action
   ```

   and Example 1 is used, *fmtmsg*() produces:

   ```
   ERROR: illegal option
   TO FIX: refer to cat in user's reference manual
   ```

APPLICATION USAGE

One or more message components may be systematically omitted from messages generated by an application by using the null value of the argument for that component.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( )

The Base Definitions volume of POSIX.1-2017, **<fmtmsg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fnmatch — match a filename string or a pathname

SYNOPSIS

#include <fnmatch.h>

int fnmatch(const char **pattern*, const char **string*, int *flags*);

DESCRIPTION

The *fnmatch*() function shall match patterns as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13.1*, *Patterns Matching a Single Character* and *Section 2.13.2*, *Patterns Matching Multiple Characters*. It checks the string specified by the *string* argument to see if it matches the pattern specified by the *pattern* argument.

The *flags* argument shall modify the interpretation of *pattern* and *string*. It is the bitwise-inclusive OR of zero or more of the flags defined in *<fnmatch.h>*. If the FNM_PATHNAME flag is set in *flags*, then a <slash> character (**'/'**) in *string* shall be explicitly matched by a <slash> in *pattern*; it shall not be matched by either the <asterisk> or <question-mark> special characters, nor by a bracket expression. If the FNM_PATHNAME flag is not set, the <slash> character shall be treated as an ordinary character.

If FNM_NOESCAPE is not set in *flags*, a <backslash> character in *pattern* followed by any other character shall match that second character in *string*. In particular, **"\\"** shall match a <backslash> in *string*. If *pattern* ends with an unescaped <backslash>, *fnmatch*() shall return a non-zero value (indicating either no match or an error). If FNM_NOESCAPE is set, a <backslash> character shall be treated as an ordinary character.

If FNM_PERIOD is set in *flags*, then a leading <period> (**'.'**) in *string* shall match a <period> in *pattern*; as described by rule 2 in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13.3*, *Patterns Used for Filename Expansion* where the location of ``leading'' is indicated by the value of FNM_PATHNAME:

* If FNM_PATHNAME is set, a <period> is ``leading'' if it is the first character in *string* or if it immediately follows a <slash>.

* If FNM_PATHNAME is not set, a <period> is ``leading'' only if it is the first character of *string*.

If FNM_PERIOD is not set, then no special restrictions are placed on matching a period.

RETURN VALUE

If *string* matches the pattern specified by *pattern*, then *fnmatch*() shall return 0. If there is no match, *fnmatch*() shall return FNM_NOMATCH, which is defined in *<fnmatch.h>*. If an error occurs, *fnmatch*() shall return another non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *fnmatch*() function has two major uses. It could be used by an application or utility that needs to read a directory and apply a pattern against each entry. The *find* utility is an example of this. It can also be used by the *pax* utility to process its *pattern* operands, or by applications that need to match strings in a similar manner.

The name *fnmatch*() is intended to imply *filename* match, rather than *pathname* match. The default action of this function is to match filename strings, rather than pathnames, since it gives no special significance to the <slash> character. With the FNM_PATHNAME flag, *fnmatch*() does match pathnames, but without tilde expansion, parameter expansion, or special treatment for a <period> at the beginning of a filename.

RATIONALE

This function replaced the REG_FILENAME flag of *regcomp*() in early proposals of this volume of POSIX.1-2017. It provides virtually the same functionality as the *regcomp*() and *regexec*() functions using the REG_FILENAME and REG_FSLASH flags (the REG_FSLASH flag was proposed for *regcomp*(), and would have had the opposite effect from FNM_PATHNAME), but with a simpler function and less system overhead.

FUTURE DIRECTIONS

None.

SEE ALSO

glob( ), *Section 2.6*, *Word Expansions*

The Base Definitions volume of POSIX.1-2017, **<fnmatch.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fopen — open a stream

SYNOPSIS

#include <stdio.h>

FILE *fopen(const char *restrict *pathname*, const char *restrict *mode*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fopen*() function shall open the file whose pathname is the string pointed to by *pathname*, and associates a stream with it.

The *mode* argument points to a string. If the string is one of the following, the file shall be opened in the indicated mode. Otherwise, the behavior is undefined.

r or *rb*	Open file for reading.
w or *wb*	Truncate to zero length or create file for writing.
a or *ab*	Append; open or create file for writing at end-of-file.
r+ or *rb*+ or *r*+*b*	Open file for update (reading and writing).
w+ or *wb*+ or *w*+*b*	Truncate to zero length or create file for update.
a+ or *ab*+ or *a*+*b*	Append; open or create file for update, writing at end-of-file.

The character **'b'** shall have no effect, but is allowed for ISO C standard conformance. Opening a file with read mode (*r* as the first character in the *mode* argument) shall fail if the file does not exist or cannot be read.

Opening a file with append mode (*a* as the first character in the *mode* argument) shall cause all subsequent writes to the file to be forced to the then current end-of-file, regardless of intervening calls to *fseek*().

When a file is opened with update mode (**'+'** as the second or third character in the *mode* argument), both input and output may be performed on the associated stream. However, the application shall ensure that output is not directly followed by input without an intervening call to *fflush*() or to a file positioning function (*fseek*(), *fsetpos*(), or *rewind*()), and input is not directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file.

When opened, a stream is fully buffered if and only if it can be determined not to refer to an interactive device. The error and end-of-file indicators for the stream shall be cleared.

If *mode* is *w*, *wb*, *a*, *ab*, *w*+, *wb*+, *w*+*b*, *a*+, *ab*+, or *a*+*b*, and the file did not previously exist, upon successful completion, *fopen*() shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last file status change and last data modification timestamps of the parent directory.

If *mode* is *w*, *wb*, *a*, *ab*, *w*+, *wb*+, *w*+*b*, *a*+, *ab*+, or *a*+*b*, and the file did not previously exist, the *fopen*() function shall create a file as if it called the *creat*() function with a value appropriate for the *path* argument interpreted from *pathname* and a value of S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH for the *mode* argument.

If *mode* is *w*, *wb*, *w*+, *wb*+, or *w*+*b*, and the file did previously exist, upon successful completion, *fopen*()

shall mark for update the last data modification and last file status change timestamps of the file.

After a successful call to the *fopen*() function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

The file descriptor associated with the opened stream shall be allocated and opened as if by a call to *open*() with the following flags:

center box tab(!); cB | cB l | l. *fopen*() Mode!*open*() Flags _ *r* or *rb*!O_RDONLY *w* or *wb*!O_WRONLY|O_CREAT|O_TRUNC *a* or *ab*!O_WRONLY|O_CREAT|O_APPEND *r*+ or *rb*+ or *r*+*b*!O_RDWR *w*+ or *wb*+ or *w*+*b*!O_RDWR|O_CREAT|O_TRUNC *a*+ or *ab*+ or *a*+*b*!O_RDWR|O_CREAT|O_APPEND

RETURN VALUE

Upon successful completion, *fopen*() shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

The *fopen*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

EINTR
A signal was caught during *fopen*().

EISDIR
The named file is a directory and *mode* requires write access.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

EMFILE
All file descriptors available to the process are currently open.

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE
The maximum allowable number of files is currently open in the system.

ENOENT
The *mode* string begins with **'r'** and a component of *pathname* does not name an existing file, or *mode* begins with **'w'** or **'a'** and a component of the path prefix of *pathname* does not name an existing file, or *pathname* is an empty string.

ENOENT or **ENOTDIR**
The *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *pathname* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory or file system that would contain the new file cannot be expanded, the file does not exist, and the file was to be created.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing

file that is neither a directory nor a symbolic link to a directory.

ENXIO
The named file is a character special or block special file, and the device associated with this special file does not exist.

EOVERFLOW
The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

EROFS
The named file resides on a read-only file system and *mode* requires write access.

The *fopen*() function may fail if:

EINVAL
The value of the *mode* argument is not valid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOMEM
Insufficient storage space is available.

ETXTBSY
The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.

The following sections are informative.

EXAMPLES

Opening a File

The following example tries to open the file named **file** for reading. The *fopen*() function returns a file pointer that is used in subsequent *fgets*() and *fclose*() calls. If the program cannot open the file, it just ignores it.

```
#include <stdio.h>
...
FILE *fp;
...
void rgrep(const char *file)
{
...
   if ((fp = fopen(file, "r")) == NULL)
       return;
...
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *creat*( ), *fclose*( ), *fdopen*( ), *fmemopen*( ), *freopen*( ), *open_memstream*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fork — create a new process

SYNOPSIS

```
#include <unistd.h>

pid_t fork(void);
```

DESCRIPTION

The *fork*() function shall create a new process. The new process (child process) shall be an exact copy of the calling process (parent process) except as detailed below:

* The child process shall have a unique process ID.
* The child process ID also shall not match any active process group ID.
* The child process shall have a different parent process ID, which shall be the process ID of the calling process.
* The child process shall have its own copy of the parent's file descriptors. Each of the child's file descriptors shall refer to the same open file description with the corresponding file descriptor of the parent.
* The child process shall have its own copy of the parent's open directory streams. Each open directory stream in the child process may share directory stream positioning with the corresponding directory stream of the parent.
* The child process shall have its own copy of the parent's message catalog descriptors.
* The child process values of *tms_utime*, *tms_stime*, *tms_cutime*, and *tms_cstime* shall be set to 0.
* The time left until an alarm clock signal shall be reset to zero, and the alarm, if any, shall be canceled; see *alarm*( ).
* All *semadj* values shall be cleared.
* File locks set by the parent process shall not be inherited by the child process.
* The set of signals pending for the child process shall be initialized to the empty set.
* Interval timers shall be reset in the child process.
* Any semaphores that are open in the parent process shall also be open in the child process.
* The child process shall not inherit any address space memory locks established by the parent process via calls to *mlockall*() or *mlock*().
* Memory mappings created in the parent shall be retained in the child process. MAP_PRIVATE mappings inherited from the parent shall also be MAP_PRIVATE mappings in the child, and any modifications to the data in these mappings made by the parent prior to calling *fork*() shall be visible to the child. Any modifications to the data in MAP_PRIVATE mappings made by the parent after *fork*() returns shall be visible only to the parent. Modifications to the data in MAP_PRIVATE mappings made by the child shall be visible only to the child.
* For the SCHED_FIFO and SCHED_RR scheduling policies, the child process shall inherit the policy and priority settings of the parent process during a *fork*() function. For other scheduling policies, the policy and priority settings on *fork*() are implementation-defined.
* Per-process timers created by the parent shall not be inherited by the child process.
* The child process shall have its own copy of the message queue descriptors of the parent. Each of the message descriptors of the child shall refer to the same open message queue description as the corresponding message descriptor of the parent.

* No asynchronous input or asynchronous output operations shall be inherited by the child process. Any use of asynchronous control blocks created by the parent produces undefined behavior.
* A process shall be created with a single thread. If a multi-threaded process calls *fork*(), the new process shall contain a replica of the calling thread and its entire address space, possibly including the states of mutexes and other resources. Consequently, to avoid errors, the child process may only execute async-signal-safe operations until such time as one of the *exec* functions is called.

  When the application calls *fork*() from a signal handler and any of the fork handlers registered by *pthread_atfork*() calls a function that is not async-signal-safe, the behavior is undefined.
* If the Trace option and the Trace Inherit option are both supported:

  If the calling process was being traced in a trace stream that had its inheritance policy set to POSIX_TRACE_INHERITED, the child process shall be traced into that trace stream, and the child process shall inherit the parent's mapping of trace event names to trace event type identifiers. If the trace stream in which the calling process was being traced had its inheritance policy set to POSIX_TRACE_CLOSE_FOR_CHILD, the child process shall not be traced into that trace stream. The inheritance policy is set by a call to the *posix_trace_attr_setinherited*() function.
* If the Trace option is supported, but the Trace Inherit option is not supported:

  The child process shall not be traced into any of the trace streams of its parent process.
* If the Trace option is supported, the child process of a trace controller process shall not control the trace streams controlled by its parent process.
* The initial value of the CPU-time clock of the child process shall be set to zero.
* The initial value of the CPU-time clock of the single thread of the child process shall be set to zero.

All other process characteristics defined by POSIX.1-2008 shall be the same in the parent and child processes. The inheritance of process characteristics not defined by POSIX.1-2008 is unspecified by POSIX.1-2008.

After *fork*(), both the parent and the child processes shall be capable of executing independently before either one terminates.

RETURN VALUE

Upon successful completion, *fork*() shall return 0 to the child process and shall return the process ID of the child process to the parent process. Both processes shall continue to execute from the *fork*() function. Otherwise, −1 shall be returned to the parent process, no child process shall be created, and *errno* shall be set to indicate the error.

ERRORS

The *fork*() function shall fail if:

EAGAIN
The system lacked the necessary resources to create another process, or the system-imposed limit on the total number of processes under execution system-wide or by a single user {CHILD_MAX} would be exceeded.

The *fork*() function may fail if:

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Many historical implementations have timing windows where a signal sent to a process group (for example, an interactive SIGINT) just prior to or during execution of *fork*() is delivered to the parent following the *fork*() but not to the child because the *fork*() code clears the child's set of pending signals. This volume of POSIX.1-2017 does not require, or even permit, this behavior. However, it is pragmatic to expect that problems of this nature may continue to exist in implementations that appear to conform to this volume of POSIX.1-2017 and pass available verification suites. This behavior is only a consequence of the implementation failing to make the interval between signal generation and delivery totally invisible. From the application's perspective, a *fork*() call should appear atomic. A signal that is generated prior to the *fork*() should be delivered prior to the *fork*(). A signal sent to the process group after the *fork*() should be delivered to both parent and child. The implementation may actually initialize internal data structures corresponding to the child's set of pending signals to include signals sent to the process group during the *fork*(). Since the *fork*() call can be considered as atomic from the application's perspective, the set would be initialized as empty and such signals would have arrived after the *fork*(); see also *<signal.h>*.

One approach that has been suggested to address the problem of signal inheritance across *fork*() is to add an **[EINTR]** error, which would be returned when a signal is detected during the call. While this is preferable to losing signals, it was not considered an optimal solution. Although it is not recommended for this purpose, such an error would be an allowable extension for an implementation.

The **[ENOMEM]** error value is reserved for those implementations that detect and distinguish such a condition. This condition occurs when an implementation detects that there is not enough memory to create the process. This is intended to be returned when **[EAGAIN]** is inappropriate because there can never be enough memory (either primary or secondary storage) to perform the operation. Since *fork*() duplicates an existing process, this must be a condition where there is sufficient memory for one such process, but not for two. Many historical implementations actually return **[ENOMEM]** due to temporary lack of memory, a case that is not generally distinct from **[EAGAIN]** from the perspective of a conforming application.

Part of the reason for including the optional error **[ENOMEM]** is because the SVID specifies it and it should be reserved for the error condition specified there. The condition is not applicable on many implementations.

IEEE Std 1003.1-1988 neglected to require concurrent execution of the parent and child of *fork*(). A system that single-threads processes was clearly not intended and is considered an unacceptable "toy implementation" of this volume of POSIX.1-2017. The only objection anticipated to the phrase "executing independently" is testability, but this assertion should be testable. Such tests require that both the parent and child can block on a detectable action of the other, such as a write to a pipe or a signal. An interactive exchange of such actions should be possible for the system to conform to the intent of this volume of POSIX.1-2017.

The **[EAGAIN]** error exists to warn applications that such a condition might occur. Whether it occurs or not is not in any practical sense under the control of the application because the condition is usually a consequence of the user's use of the system, not of the application's code. Thus, no application can or should rely upon its occurrence under any circumstances, nor should the exact semantics of what concept of "user" is used be of concern to the application developer. Validation writers should be cognizant of this limitation.

There are two reasons why POSIX programmers call *fork*(). One reason is to create a new thread of control within the same program (which was originally only possible in POSIX by creating a new process); the other is to create a new process running a different program. In the latter case, the call to *fork*() is soon followed by a call to one of the *exec* functions.

The general problem with making *fork*() work in a multi-threaded world is what to do with all of the threads. There are two alternatives. One is to copy all of the threads into the new process. This causes the programmer or implementation to deal with threads that are suspended on system calls or that might be about to execute system calls that should not be executed in the new process. The other alternative is to copy only the thread that calls *fork*(). This creates the difficulty that the state of process-local resources is usually held in process memory. If a thread that is not calling *fork*() holds a resource, that resource is never released in the child process because the thread whose job it is to release the resource does not exist in the child process.

When a programmer is writing a multi-threaded program, the first described use of *fork*(), creating new threads in the same program, is provided by the *pthread_create*() function. The *fork*() function is thus used only to run new programs, and the effects of calling functions that require certain resources between the call to *fork*() and the call to an *exec* function are undefined.

The addition of the *forkall*() function to the standard was considered and rejected. The *forkall*() function lets all the threads in the parent be duplicated in the child. This essentially duplicates the state of the parent in the child. This allows threads in the child to continue processing and allows locks and the state to be preserved without explicit *pthread_atfork*() code. The calling process has to ensure that the threads processing state that is shared between the parent and child (that is, file descriptors or MAP_SHARED memory) behaves properly after *forkall*(). For example, if a thread is reading a file descriptor in the parent when *forkall*() is called, then two threads (one in the parent and one in the child) are reading the file descriptor after the *forkall*(). If this is not desired behavior, the parent process has to synchronize with such threads before calling *forkall*().

While the *fork*() function is async-signal-safe, there is no way for an implementation to determine whether the fork handlers established by *pthread_atfork*() are async-signal-safe. The fork handlers may attempt to execute portions of the implementation that are not async-signal-safe, such as those that are protected by mutexes, leading to a deadlock condition. It is therefore undefined for the fork handlers to execute functions that are not async-signal-safe when *fork*() is called from a signal handler.

When *forkall*() is called, threads, other than the calling thread, that are in functions that can return with an **[EINTR]** error may have those functions return **[EINTR]** if the implementation cannot ensure that the function behaves correctly in the parent and child. In particular, *pthread_cond_wait*() and *pthread_cond_timedwait*() need to return in order to ensure that the condition has not changed. These functions can be awakened by a spurious condition wakeup rather than returning **[EINTR]**.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fcntl*( ), *posix_trace_attr_getinherited*( ), *posix_trace_eventid_equal*( ), *pthread_atfork*( ), *semop*( ), *signal*( ), *times*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fpathconf, pathconf — get configurable pathname variables

SYNOPSIS

#include <unistd.h>

long fpathconf(int *fildes*, int *name*);
long pathconf(const char **path*, int *name*);

DESCRIPTION

The *fpathconf*() and *pathconf*() functions shall determine the current value of a configurable limit or option (*variable*) that is associated with a file or directory.

For *pathconf*(), the *path* argument points to the pathname of a file or directory.

For *fpathconf*(), the *fildes* argument is an open file descriptor.

The *name* argument represents the variable to be queried relative to that file or directory. Implementations shall support all of the variables listed in the following table and may support others. The variables in the following table come from *<limits.h>* or *<unistd.h>* and the symbolic constants, defined in *<unistd.h>*, are the corresponding values used for *name*.

center box tab(!); cB | cB | cB l | l | l. Variable!Value of *name*!Requirements _ {FILESIZEBITS}!_PC_FILESIZEBITS!4, 7 {LINK_MAX}!_PC_LINK_MAX!1 {MAX_CANON}!_PC_MAX_CANON!2 {MAX_INPUT}!_PC_MAX_INPUT!2 {NAME_MAX}!_PC_NAME_MAX!3, 4 {PATH_MAX}!_PC_PATH_MAX!4, 5 {PIPE_BUF}!_PC_PIPE_BUF!6 {POSIX2_SYMLINKS}!_PC_2_SYMLINKS!4 {POSIX_ALLOC_SIZE_MIN}!_PC_ALLOC_SIZE_MIN!10 {POSIX_REC_INCR_XFER_SIZE}!_PC_REC_INCR_XFER_SIZE!10 {POSIX_REC_MAX_XFER_SIZE}!_PC_REC_MAX_XFER_SIZE!10 {POSIX_REC_MIN_XFER_SIZE}!_PC_REC_MIN_XFER_SIZE!10 {POSIX_REC_XFER_ALIGN}!_PC_REC_XFER_ALIGN!10 {SYMLINK_MAX}!_PC_SYMLINK_MAX!4, 9 _POSIX_CHOWN_RESTRICTED!_PC_CHOWN_RESTRICTED!7 _POSIX_NO_TRUNC!_PC_NO_TRUNC!3, 4 _POSIX_VDISABLE!_PC_VDISABLE!2 _POSIX_ASYNC_IO!_PC_ASYNC_IO!8 _POSIX_PRIO_IO!_PC_PRIO_IO!8 _POSIX_SYNC_IO!_PC_SYNC_IO!8 _POSIX_TIMESTAMP_RESOLUTION!_PC_TIMESTAMP_RESOLUTION!1

Requirements

1. If *path* or *fildes* refers to a directory, the value returned shall apply to the directory itself.
2. If *path* or *fildes* does not refer to a terminal file, it is unspecified whether an implementation supports an association of the variable name with the specified file.
3. If *path* or *fildes* refers to a directory, the value returned shall apply to filenames within the directory.
4. If *path* or *fildes* does not refer to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
5. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of a relative pathname that would not cross any mount points when the specified directory is the working directory.
6. If *path* refers to a FIFO, or *fildes* refers to a pipe or FIFO, the value returned shall apply to the referenced object. If *path* or *fildes* refers to a directory, the value returned shall apply to any FIFO that exists or can be created within the directory. If *path* or *fildes* refers to any other type of file, it is unspecified whether an implementation supports an association of the variable name with the specified file.

7. If *path* or *fildes* refers to a directory, the value returned shall apply to any files, other than directories, that exist or can be created within the directory.
8. If *path* or *fildes* refers to a directory, it is unspecified whether an implementation supports an association of the variable name with the specified file.
9. If *path* or *fildes* refers to a directory, the value returned shall be the maximum length of the string that a symbolic link in that directory can contain.
10. If *path* or *fildes* des does not refer to a regular file, it is unspecified whether an implementation supports an association of the variable name with the specified file. If an implementation supports such an association for other than a regular file, the value returned is unspecified.

RETURN VALUE

If *name* is an invalid value, both *pathconf*() and *fpathconf*() shall return −1 and set *errno* to indicate the error.

If the variable corresponding to *name* is described in <*limits.h*> as a maximum or minimum value and the variable has no limit for the path or file descriptor, both *pathconf*() and *fpathconf*() shall return −1 without changing *errno*. Note that indefinite limits do not imply infinite limits; see <*limits.h*>.

If the implementation needs to use *path* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *path*, or if the process did not have appropriate privileges to query the file specified by *path*, or *path* does not exist, *pathconf*() shall return −1 and set *errno* to indicate the error.

If the implementation needs to use *fildes* to determine the value of *name* and the implementation does not support the association of *name* with the file specified by *fildes*, or if *fildes* is an invalid file descriptor, *fpathconf*() shall return −1 and set *errno* to indicate the error.

Otherwise, *pathconf*() or *fpathconf*() shall return the current variable value for the file or directory without changing *errno*. The value returned shall not be more restrictive than the corresponding value available to the application when it was compiled with the implementation's <*limits.h*> or <*unistd.h*>.

If the variable corresponding to *name* is dependent on an unsupported option, the results are unspecified.

ERRORS

The *pathconf*() function shall fail if:

EINVAL
The value of *name* is not valid.

EOVERFLOW
The value of *name* is _PC_TIMESTAMP_RESOLUTION and the resolution is larger than {LONG_MAX}.

The *pathconf*() function may fail if:

EACCES
Search permission is denied for a component of the path prefix.

EINVAL
The implementation does not support an association of the variable *name* with the specified file.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *fpathconf*() function shall fail if:

EINVAL
The value of *name* is not valid.

EOVERFLOW
The value of *name* is _PC_TIMESTAMP_RESOLUTION and the resolution is larger than {LONG_MAX}.

The *fpathconf*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINVAL
The implementation does not support an association of the variable *name* with the specified file.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Application developers should check whether an option, such as _POSIX_ADVISORY_INFO, is supported prior to obtaining and using values for related variables such as {POSIX_ALLOC_SIZE_MIN}.

RATIONALE

The *pathconf*() function was proposed immediately after the *sysconf*() function when it was realized that some configurable values may differ across file system, directory, or device boundaries.

For example, {NAME_MAX} frequently changes between System V and BSD-based file systems; System V uses a maximum of 14, BSD 255. On an implementation that provides both types of file systems, an application would be forced to limit all pathname components to 14 bytes, as this would be the value specified in *<limits.h>* on such a system.

Therefore, various useful values can be queried on any pathname or file descriptor, assuming that appropriate privileges are in place.

The value returned for the variable {PATH_MAX} indicates the longest relative pathname that could be given if the specified directory is the current working directory of the process. A process may not always be able to generate a name that long and use it if a subdirectory in the pathname crosses into a more restrictive file system. Note that implementations are allowed to accept pathnames longer than {PATH_MAX} bytes long, but are not allowed to return pathnames longer than this unless the user specifies a larger buffer using a function that provides a buffer size argument.

The value returned for the variable _POSIX_CHOWN_RESTRICTED also applies to directories that do not have file systems mounted on them. The value may change when crossing a mount point, so applications that need to know should check for each directory. (An even easier check is to try the *chown*() function and look for an error in case it happens.)

Unlike the values returned by *sysconf*(), the pathname-oriented variables are potentially more volatile and are not guaranteed to remain constant throughout the lifetime of the process. For example, in between two calls to *pathconf*(), the file system in question may have been unmounted and remounted with different characteristics.

Also note that most of the errors are optional. If one of the variables always has the same value on an implementation, the implementation need not look at *path* or *fildes* to return that value and is, therefore, not required to detect any of the errors except the meaning of **[EINVAL]** that indicates that the value of *name* is not valid for that variable, and the **[EOVERFLOW]** error that indicates the value to be returned is larger than {LONG_MAX}.

If the value of any of the limits is unspecified (logically infinite), they will not be defined in *<limits.h>* and the *pathconf*() and *fpathconf*() functions return −1 without changing *errno*. This can be distinguished from the case of giving an unrecognized *name* argument because *errno* is set to **[EINVAL]** in this case.

Since −1 is a valid return value for the *pathconf*() and *fpathconf*() functions, applications should set *errno* to zero before calling them and check *errno* only if the return value is −1.

For the case of {SYMLINK_MAX}, since both *pathconf*() and *open*() follow symbolic links, there is no way that *path* or *fildes* could refer to a symbolic link.

It was the intention of IEEE Std 1003.1d-1999 that the following variables:

{POSIX_ALLOC_SIZE_MIN}	{POSIX_REC_INCR_XFER_SIZE}
{POSIX_REC_MAX_XFER_SIZE}	{POSIX_REC_MIN_XFER_SIZE}
{POSIX_REC_XFER_ALIGN}	

only applied to regular files, but Note 10 also permits implementation of the advisory semantics on other file types unique to an implementation (for example, a character special device).

The **[EOVERFLOW]** error for _PC_TIMESTAMP_RESOLUTION cannot occur on POSIX-compliant file systems because POSIX requires a timestamp resolution no larger than one second. Even on 32-bit systems, this can be represented without overflow.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *confstr*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<unistd.h>**

The Shell and Utilities volume of POSIX.1-2017, *getconf*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fpclassify — classify real floating type

SYNOPSIS

```
#include <math.h>

int fpclassify(real-floating x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fpclassify*() macro shall classify its argument value as NaN, infinite, normal, subnormal, zero, or into another implementation-defined category. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then classification is based on the type of the argument.

RETURN VALUE

The *fpclassify*() macro shall return the value of the number classification macro appropriate to the value of its argument.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isfinite( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

dprintf, fprintf, printf, snprintf, sprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>

int dprintf(int fildes, const char *restrict format, ...);
int fprintf(FILE *restrict stream, const char *restrict format, ...);
int printf(const char *restrict format, ...);
int snprintf(char *restrict s, size_t n,
    const char *restrict format, ...);
int sprintf(char *restrict s, const char *restrict format, ...);
```

DESCRIPTION

Excluding *dprintf*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fprintf*() function shall place output on the named output *stream*. The *printf*() function shall place output on the standard output stream *stdout*. The *sprintf*() function shall place output followed by the null byte, **'\0'**, in consecutive bytes starting at **s*; it is the user's responsibility to ensure that enough space is available.

The *dprintf*() function shall be equivalent to the *fprintf*() function, except that *dprintf*() shall write output to the file associated with the file descriptor specified by the *fildes* argument rather than place output on a stream.

The *snprintf*() function shall be equivalent to *sprintf*(), with the addition of the *n* argument which states the size of the buffer referred to by *s*. If *n* is zero, nothing shall be written and *s* may be a null pointer. Otherwise, output bytes beyond the *n*-1st shall be discarded instead of being written to the array, and a null byte is written at the end of the bytes actually written into the array.

If copying takes place between objects that overlap as a result of a call to *sprintf*() or *snprintf*(), the results are undefined.

Each of these functions converts, formats, and prints its arguments under control of the *format*. The *format* is a character string, beginning and ending in its initial shift state, if any. The *format* is composed of zero or more directives: *ordinary characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments shall be evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character **%** (see below) is replaced by the sequence "%*n*$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument conversion specifications (that is, "%*n*$" and "**m*$"), or unnumbered argument conversion specifications (that is, **%** and *****), but not both. The only exception to this is that **%%** can be mixed with the "%*n*$" form. The results of mixing numbered and unnumbered argument specifications in a *format* string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N−1*)th, are specified in the format string.

In format strings containing the "%*n*$" form of conversion specification, numbered arguments in the argument list can be referenced from the format string as many times as required.

In format strings containing the **%** form of conversion specification, each conversion specification uses the first unused argument in the argument list.

All forms of the *fprintf*() functions allow for the insertion of a language-dependent radix character in the output string. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (**'.'**).

Each conversion specification is introduced by the **'%'** character or by the character sequence "%*n*$", after which the following appear in sequence:

* Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
* An optional minimum *field width*. If the converted value has fewer bytes than the field width, it shall be padded with <space> characters by default on the left; it shall be padded on the right if the left-adjustment flag (**'−'**), described below, is given to the field width. The field width takes the form of an <asterisk> (**'*'**), described below, or a decimal integer.
* An optional *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversion specifiers; the number of digits to appear after the radix character for the **a**, **A**, **e**, **E**, **f**, and **F** conversion specifiers; the maximum number of significant digits for the **g** and **G** conversion specifiers; or the maximum number of bytes to be printed from a string in the **s** and **S** conversion specifiers. The precision takes the form of a <period> (**'.'**) followed either by an <asterisk> (**'*'**), described below, or an optional decimal digit string, where a null digit string is treated as zero. If a precision appears with any other conversion specifier, the behavior is undefined.
* An optional length modifier that specifies the size of the argument.
* A *conversion specifier* character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an <asterisk> (**'*'**). In this case an argument of type **int** supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a **'−'** flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In *format* strings containing the "%*n*$" form of a conversion specification, a field width or precision may be indicated by the sequence "**m*$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:

```
printf("%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag characters and their meanings are:

'	(The <apostrophe>.) The integer portion of the result of a decimal conversion (**%i**, **%d**, **%u**, **%f**, **%F**, **%g**, or **%G**) shall be formatted with thousands' grouping characters. For other conversions the behavior is undefined. The non-monetary grouping character is used.
−	The result of the conversion shall be left-justified within the field. The conversion is right-justified if this flag is not specified.
+	The result of a signed conversion shall always begin with a sign (**'+'** or **'−'**). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.
<space>	If the first character of a signed conversion is not a sign or if a signed conversion results in no characters, a <space> shall be prefixed to the result. This means that if the <space> and **'+'** flags both appear, the <space> flag shall be ignored.
#	Specifies that the value is to be converted to an alternative form. For **o** conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be a zero (if the value and precision are both 0, a single 0 is printed). For **x** or **X** conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, the result shall always contain a radix character, even if no digits follow the radix character. Without

this flag, a radix character appears in the result of these conversions only if a digit follows it. For **g** and **G** conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

0 For **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the **'0'** and **'−'** flags both appear, the **'0'** flag is ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversion specifiers, if a precision is specified, the **'0'** flag shall be ignored. If the **'0'** and <apostrophe> flags both appear, the grouping characters are inserted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

hh Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following **n** conversion specifier applies to a pointer to a **signed char** argument.

h Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **short** or **unsigned short** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short** or **unsigned short** before printing); or that a following **n** conversion specifier applies to a pointer to a **short** argument.

l (ell) Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long** or **unsigned long** argument; that a following **n** conversion specifier applies to a pointer to a **long** argument; that a following **c** conversion specifier applies to a **wint_t** argument; that a following **s** conversion specifier applies to a pointer to a **wchar_t** argument; or has no effect on a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier.

ll (ell-ell)
Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long long** or **unsigned long long** argument; or that a following **n** conversion specifier applies to a pointer to a **long long** argument.

j Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to an **intmax_t** or **uintmax_t** argument; or that a following **n** conversion specifier applies to a pointer to an **intmax_t** argument.

z Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **size_t** or the corresponding signed integer type argument; or that a following **n** conversion specifier applies to a pointer to a signed integer type corresponding to a **size_t** argument.

t Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **ptrdiff_t** or the corresponding **unsigned** type argument; or that a following **n** conversion specifier applies to a pointer to a **ptrdiff_t** argument.

L Specifies that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to a **long double** argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The conversion specifiers and their meanings are:

d, i The **int** argument shall be converted to a signed decimal in the style "[−]*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

o The **unsigned** argument shall be converted to unsigned octal format in the style "*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

u The **unsigned** argument shall be converted to unsigned decimal format in the style "*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

x The **unsigned** argument shall be converted to unsigned hexadecimal format in the style "*dddd*"; the letters **"abcdef"** are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision is 1. The result of converting zero with an explicit precision of zero shall be no characters.

X Equivalent to the **x** conversion specifier, except that letters **"ABCDEF"** are used instead of **"abcdef"**.

f, F The **double** argument shall be converted to decimal notation in the style "[−]*ddd.ddd*", where the number of digits after the radix character is equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no **'#'** flag is present, no radix character shall appear. If a radix character appears, at least one digit appears before it. The low-order digit shall be rounded in an implementation-defined manner.

A **double** argument representing an infinity shall be converted in one of the styles **"[-]inf"** or **"[-]infinity"**; which style is implementation-defined. A **double** argument representing a NaN shall be converted in one of the styles "[−]nan(*n-char-sequence*)" or **"[-]nan"**; which style, and the meaning of any *n-char-sequence*, is implementation-defined. The **F** conversion specifier produces **"INF"**, **"INFINITY"**, or **"NAN"** instead of **"inf"**, **"infinity"**, or **"nan"**, respectively.

e, E The **double** argument shall be converted in the style "[−]*d.ddd*e±*dd*", where there is one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it is equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no **'#'** flag is present, no radix character shall appear. The low-order digit shall be rounded in an implementation-defined manner. The **E** conversion specifier shall produce a number with **'E'** instead of **'e'** introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

g, G The **double** argument representing a floating-point number shall be converted in the style **f** or **e** (or in the style **F** or **E** in the case of a **G** conversion specifier), depending on the value converted and the precision. Let **P** equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style **E** would have an exponent of *X*:

-- If **P**>*X*≥−4, the conversion shall be with style **f** (or **F**) and precision **P**−(*X*+1).

-- Otherwise, the conversion shall be with style **e** (or **E**) and precision **P**−1.

Finally, unless the **'#'** flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

a, A A **double** argument representing a floating-point number shall be converted in the style "[−]0x*h.hhhh*p±*d*", where there is one hexadecimal digit (which shall be non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point character and the number of hexadecimal digits after it is equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type **double**, except that trailing zeros may be omitted; if the precision is zero and the **'#'** flag is not specified, no decimal-point character shall appear. The letters **"abcdef"** shall be used for **a** conversion and the letters **"ABCDEF"** for **A** conversion. The **A** conversion specifier produces a number with **'X'** and **'P'** instead of **'x'** and **'p'**. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

c The **int** argument shall be converted to an **unsigned char**, and the resulting byte shall be written.

If an **l** (ell) qualifier is present, the **wint_t** argument shall be converted as if by an **ls** conversion specification with no precision and an argument that points to a two-element array of type **wchar_t**, the first element of which contains the **wint_t** argument to the **ls** conversion specification and the second element contains a null wide character.

s The argument shall be a pointer to an array of **char**. Bytes from the array shall be written up to (but not including) any terminating null byte. If the precision is specified, no more than that many bytes shall be written. If the precision is not specified or is greater than the size of the array, the application shall ensure that the array contains a null byte.

If an **l** (ell) qualifier is present, the argument shall be a pointer to an array of type **wchar_t**. Wide characters from the array shall be converted to characters (each as if by a call to the *wcrtomb*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted) up to and including a terminating null wide character. The resulting characters shall be written up to (but not including) the terminating null character (byte). If no precision is specified, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many characters (bytes) shall be written (including shift sequences, if any), and the array shall contain a null wide character if, to equal the character sequence length given by the precision, the function would need to access a wide character one past the end of the array. In no case shall a partial character be written.

p The argument shall be a pointer to **void**. The value of the pointer is converted to a sequence of printable characters, in an implementation-defined manner.

n The argument shall be a pointer to an integer into which is written the number of bytes written to the output so far by this call to one of the *fprintf*() functions. No argument is converted.

C Equivalent to **lc**.

S Equivalent to **ls**.

% Print a **'%'** character; no argument is converted. The complete conversion specification shall be **%%**.

If a conversion specification does not match one of the above forms, the behavior is undefined. If any argument is not the correct type for the corresponding conversion specification, the behavior is undefined.

In no case shall a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fprintf*() and *printf*() are printed as if *fputc*() had been called.

For the **a** and **A** conversion specifiers, if FLT_RADIX is a power of 2, the value shall be correctly rounded to a hexadecimal floating number with the given precision.

For **a** and **A** conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For the **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

The last data modification and last file status change timestamps of the file shall be marked for update:

1. Between the call to a successful execution of *fprintf*() or *printf*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*()
2. Upon successful completion of a call to *dprintf*()

RETURN VALUE

Upon successful completion, the *dprintf*(), *fprintf*(), and *printf*() functions shall return the number of bytes transmitted.

Upon successful completion, the *sprintf*() function shall return the number of bytes written to *s*, excluding the terminating null byte.

Upon successful completion, the *snprintf*() function shall return the number of bytes that would be written to *s* had *n* been sufficiently large excluding the terminating null byte.

If an output error was encountered, these functions shall return a negative value and set *errno* to indicate the error.

If the value of *n* is zero on a call to *snprintf*(), nothing shall be written, the number of bytes that would have been written had *n* been sufficiently large excluding the terminating null shall be returned, and *s* may be a null pointer.

ERRORS

For the conditions under which *dprintf*(), *fprintf*(), and *printf*() fail and may fail, refer to *fputc*( ) or *fputwc*( ).

In addition, all forms of *fprintf*() shall fail if:

EILSEQ
A wide-character code that does not correspond to a valid character has been detected.

EOVERFLOW
The value to be returned is greater than {INT_MAX}.

The *dprintf*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

The *dprintf*(), *fprintf*(), and *printf*() functions may fail if:

ENOMEM
Insufficient storage space is available.

The *snprintf*() function shall fail if:

EOVERFLOW
The value of *n* is greater than {INT_MAX}.

The following sections are informative.

EXAMPLES

Printing Language-Independent Date and Time

The following statement can be used to print date and time using a language-independent format:

```
printf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the following string:

```
"%s, %s %d, %d:%.2d\n"
```

This example would produce the following message:

```
Sunday, July 3, 10:02
```

For German usage, *format* could be a pointer to the following string:

```
"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

This definition of *format* would produce the following message:

```
Sonntag, 3. Juli, 10:02
```

Printing File Information

The following example prints information about the type, permissions, and number of links of a specific file in a directory.

The first two calls to *printf*() use data decoded from a previous *stat*() call. The user-defined *strperm*() function shall return a string similar to the one at the beginning of the output for the following command:

```
ls -l
```

The next call to *printf*() outputs the owner's name if it is found using *getpwuid*(); the *getpwuid*() function shall return a **passwd** structure from which the name of the user is extracted. If the user name is not found, the program instead prints out the numeric value of the user ID.

The next call prints out the group name if it is found using *getgrgid*(); *getgrgid*() is very similar to *getpwuid*() except that it shall return group information based on the group number. Once again, if the group is not found, the program prints the numeric value of the group for the entry.

The final call to *printf*() prints the size of the file.

```
#include <stdio.h>
#include <sys/types.h>
#include <pwd.h>
#include <grp.h>

char *strperm (mode_t);
...
struct stat statbuf;
struct passwd *pwd;
struct group *grp;
...
printf("%10.10s", strperm (statbuf.st_mode));
printf("%4d", statbuf.st_nlink);

if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
    printf(" %-8.8s", pwd->pw_name);
else
    printf(" %-8ld", (long) statbuf.st_uid);

if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8ld", (long) statbuf.st_gid);

printf("%9jd", (intmax_t) statbuf.st_size);
```

```
...
```

Printing a Localized Date String

The following example gets a localized date string. The *nl_langinfo*() function shall return the localized date string, which specifies the order and layout of the date. The *strftime*() function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*. The *printf*() function then outputs *datestring* and the name of the entry.

```
#include <stdio.h>
#include <time.h>
#include <langinfo.h>
...
struct dirent *dp;
struct tm *tm;
char datestring[256];
...
strftime(datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);

printf(" %s %s\n", datestring, dp->d_name);
...
```

Printing Error Information

The following example uses *fprintf*() to write error information to standard error.

In the first group of calls, the program tries to open the password lock file named **LOCKFILE**. If the file already exists, this is an error, as indicated by the O_EXCL flag on the *open*() function. If the call fails, the program assumes that someone else is updating the password file, and the program exits.

The next group of calls saves a new password file as the current password file by creating a link between **LOCKFILE** and the new password file **PASSWDFILE**.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
...
int pfd;
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
    S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}
...
if (link(LOCKFILE,PASSWDFILE) == -1) {
    fprintf(stderr, "Link error: %s\n", strerror(errno));
    exit(1);
}
```

```
...
```

Printing Usage Information

The following example checks to make sure the program has the necessary arguments, and uses *fprintf*() to print usage information if the expected number of arguments is not present.

```
#include <stdio.h>
#include <stdlib.h>
...
char *Options = "hdbtl";
...
if (argc < 2) {
    fprintf(stderr, "Usage: %s -%s <file\n", argv[0], Options); exit(1);
}
...
```

Formatting a Decimal String

The following example prints a key and data pair on *stdout*. Note use of the <asterisk> (**'*'**) in the format string; this ensures the correct number of decimal places for the element based on the number of elements requested.

```
#include <stdio.h>
...
long i;
char *keystr;
int elementlen, len;
...
while (len < elementlen) {
...
    printf("%s Element%0*ld\n", keystr, elementlen, i);
...
}
```

Creating a Pathname

The following example creates a pathname using information from a previous *getpwnam*() function that returned the password database entry of the user.

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>
...
char *pathname;
struct passwd *pw;
size_t len;
...
// digits required for pid_t is number of bits times
// log2(10) = approx 10/33
len = strlen(pw->pw_dir) + 1 + 1+(sizeof(pid_t)*80+32)/33 +
    sizeof ".out";
pathname = malloc(len);
```

```
if (pathname != NULL)
{
    snprintf(pathname, len, "%s/%jd.out", pw->pw_dir,
        (intmax_t)getpid());
    ...
}
```

Reporting an Event

The following example loops until an event has timed out. The *pause*() function waits forever unless it receives a signal. The *fprintf*() statement should never occur due to the possible return values of *pause*().

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <errno.h>
...
while (!event_complete) {
...
    if (pause() != -1 || errno != EINTR)
        fprintf(stderr, "pause: unknown error: %s\n", strerror(errno));
}
...
```

Printing Monetary Information

The following example uses *strfmon*() to convert a number and store it as a formatted monetary string named *convbuf*. If the first number is printed, the program prints the format and the description; otherwise, it just prints the number.

```
#include <monetary.h>
#include <stdio.h>
...
struct tblfmt {
    char *format;
    char *description;
};

struct tblfmt table[] = {
    { "%n", "default formatting" },
    { "%11n", "right align within an 11 character field" },
    { "%#5n", "aligned columns for values up to 99 999" },
    { "%=*#5n", "specify a fill character" },
    { "%=0#5n", "fill characters do not use grouping" },
    { "%^#5n", "disable the grouping separator" },
    { "%^#5.0n", "round off to whole units" },
    { "%^#5.4n", "increase the precision" },
    { "%(#5n", "use an alternative pos/neg style" },
    { "%!(#5n", "disable the currency symbol" },
};
...
float input[3];
int i, j;
char convbuf[100];
...
strfmon(convbuf, sizeof(convbuf), table[i].format, input[j]);
```

```
if (j == 0) {
    printf("%s%s%s\n", table[i].format,
        convbuf, table[i].description);
}
else {
    printf("%s\n", convbuf);
}
...
```

Printing Wide Characters

The following example prints a series of wide characters. Suppose that **"L'@'"** expands to three bytes:

```
wchar_t wz [3] = L"@@";       // Zero-terminated
wchar_t wn [3] = L"@@@";      // Unterminated

fprintf (stdout,"%ls", wz);   // Outputs 6 bytes
fprintf (stdout,"%ls", wn);   // Undefined because wn has no terminator
fprintf (stdout,"%4ls", wz);  // Outputs 3 bytes
fprintf (stdout,"%4ls", wn);  // Outputs 3 bytes; no terminator needed
fprintf (stdout,"%9ls", wz);  // Outputs 6 bytes
fprintf (stdout,"%9ls", wn);  // Outputs 9 bytes; no terminator needed
fprintf (stdout,"%10ls", wz); // Outputs 6 bytes
fprintf (stdout,"%10ls", wn); // Undefined because wn has no terminator
```

In the last line of the example, after processing three characters, nine bytes have been output. The fourth character must then be examined to determine whether it converts to one byte or more. If it converts to more than one byte, the output is only nine bytes. Since there is no fourth character in the array, the behavior is undefined.

APPLICATION USAGE

If the application calling *fprintf*() has any objects of type **wint_t** or **wchar_t**, it must also include the *<wchar.h>* header to have these objects defined.

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fputc*( ), *fscanf*( ), *setlocale*( ), *strfmon*( ), *wcrtomb*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fputc — put a byte on a stream

SYNOPSIS

#include <stdio.h>

int fputc(int *c*, FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fputc*() function shall write the byte specified by *c* (converted to an **unsigned char**) to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and shall advance the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the byte shall be appended to the output stream.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputc*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

RETURN VALUE

Upon successful completion, *fputc*() shall return the value it has written. Otherwise, it shall return EOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

The *fputc*() function shall fail if either the *stream* is unbuffered or the *stream*'s buffer needs to be flushed, and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying *stream* is not a valid file descriptor open for writing.

EFBIG
An attempt was made to write to a file that exceeds the maximum file size.

EFBIG
An attempt was made to write to a file that exceeds the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOSPC
There was no free space remaining on the device containing the file.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fputc*() function may fail if:

ENOMEM
Insufficient storage space is available.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *ferror*( ), *fopen*( ), *getrlimit*( ), *putc*( ), *puts*( ), *setbuf*( ), *ulimit*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fputs — put a string on a stream

SYNOPSIS

#include <stdio.h>

int fputs(const char *restrict *s*, FILE *restrict *stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fputs*() function shall write the null-terminated string pointed to by *s* to the stream pointed to by *stream*. The terminating null byte shall not be written.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputs*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

RETURN VALUE

Upon successful completion, *fputs*() shall return a non-negative number. Otherwise, it shall return EOF, set an error indicator for the stream, and set *errno* to indicate the error.

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

Printing to Standard Output

The following example gets the current time, converts it to a string using *localtime*() and *asctime*(), and prints it to standard output using *fputs*(). It then prints the number of minutes to an event for which it is waiting.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
fputs(asctime(localtime(&now)), stdout);
printf("There are still %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

The *puts*() function appends a <newline> while *fputs*() does not.

This volume of POSIX.1-2017 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

* Return a constant value.

* Return the last character written.
* Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards-compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

The *fputs*() function is one whose source code was specified in the referenced *The C Programming Language*. In the original edition, the function had no defined return value, yet many practical implementations would, as a side-effect, return the value of the last character written as that was the value remaining in the accumulator used as a return value. In the second edition of the book, either the fixed value 0 or EOF would be returned depending upon the return value of *ferror*(); however, for compatibility with extant implementations, several implementations would, upon success, return a positive value representing the last byte written.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *putc*( ), *puts*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fputwc — put a wide-character code on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t fputwc(wchar_t wc, FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fputwc*() function shall write the character corresponding to the wide-character code *wc* to the output stream pointed to by *stream*, at the position indicated by the associated file-position indicator for the stream (if defined), and advances the indicator appropriately. If the file cannot support positioning requests, or if the stream was opened with append mode, the character is appended to the output stream. If an error occurs while writing the character, the shift state of the output file is left in an undefined state.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputwc*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

The *fputwc*() function shall not change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, *fputwc*() shall return *wc*. Otherwise, it shall return WEOF, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

The *fputwc*() function shall fail if either the stream is unbuffered or data in the *stream*'s buffer needs to be written, and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor underlying *stream* and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying *stream* is not a valid file descriptor open for writing.

EFBIG
An attempt was made to write to a file that exceeds the maximum file size or the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EILSEQ
The wide-character code *wc* does not correspond to a valid character.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOSPC
There was no free space remaining on the device containing the file.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process. A SIGPIPE signal shall also be sent to the thread.

The *fputwc*() function may fail if:

ENOMEM
Insufficient storage space is available.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *ferror*( ), *fopen*( ), *setbuf*( ), *ulimit*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fputws — put a wide-character string on a stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fputws(const wchar_t *restrict ws, FILE *restrict stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fputws*() function shall write a character string corresponding to the (null-terminated) wide-character string pointed to by *ws* to the stream pointed to by *stream*. No character corresponding to the terminating null wide-character code shall be written.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fputws*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

RETURN VALUE

Upon successful completion, *fputws*() shall return a non-negative number. Otherwise, it shall return −1, set an error indicator for the stream, and set *errno* to indicate the error.

ERRORS

Refer to *fputwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *fputws*() function does not append a <newline>.

This volume of POSIX.1-2017 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

* Return a constant value.
* Return the last character written.
* Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards-compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fread — binary input

SYNOPSIS

```
#include <stdio.h>

size_t fread(void *restrict ptr, size_t size, size_t nitems,
   FILE *restrict stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fread*() function shall read into the array pointed to by *ptr* up to *nitems* elements whose size is specified by *size* in bytes, from the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fgetc*() function and the results stored, in the order read, in an array of **unsigned char** exactly overlaying the object. The file position indicator for the stream (if defined) shall be advanced by the number of bytes successfully read. If an error occurs, the resulting value of the file position indicator for the stream is unspecified. If a partial element is read, its value is unspecified.

The *fread*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *fscanf*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, *fread*() shall return the number of elements successfully read which is less than *nitems* only if a read error or end-of-file is encountered. If *size* or *nitems* is 0, *fread*() shall return 0 and the contents of the array and the state of the stream remain unchanged. Otherwise, if a read error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fgetc*( ).

The following sections are informative.

EXAMPLES

Reading from a Stream

The following example transfers a single 100-byte fixed length record from the *fp* stream into the array pointed to by *buf*.

```
#include <stdio.h>
...
size_t elements_read;
char buf[100];
FILE *fp;
...
elements_read = fread(buf, sizeof(buf), 1, fp);
...
```

If a read error occurs, *elements_read* will be zero but the number of bytes read from the stream could be anything from zero to *sizeof*(*buf*)−1.

The following example reads multiple single-byte elements from the *fp* stream into the array pointed to by *buf*.

```
#include <stdio.h>
...
size_t bytes_read;
char buf[100];
FILE *fp;
...
bytes_read = fread(buf, 1, sizeof(buf), fp);
...
```

If a read error occurs, *bytes_read* will contain the number of bytes read from the stream.

APPLICATION USAGE

The *ferror*() or *feof*() functions must be used to distinguish between an error condition and an end-of-file condition.

Because of possible differences in element length and byte ordering, files written using *fwrite*() are application-dependent, and possibly cannot be read using *fread*() by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *feof*( ), *ferror*( ), *fgetc*( ), *fopen*( ), *fscanf*( ), *getc*( ), *gets*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

free — free allocated memory

SYNOPSIS

```
#include <stdlib.h>

void free(void *ptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *free*() function shall cause the space pointed to by *ptr* to be deallocated; that is, made available for further allocation. If *ptr* is a null pointer, no action shall occur. Otherwise, if the argument does not match a pointer earlier returned by a function in POSIX.1-2008 that allocates memory as if by *malloc*(), or if the space has been deallocated by a call to *free*() or *realloc*(), the behavior is undefined.

Any use of a pointer that refers to freed space results in undefined behavior.

RETURN VALUE

The *free*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

There is now no requirement for the implementation to support the inclusion of *<malloc.h>*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

calloc( ), *malloc*( ), *posix_memalign*( ), *realloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

freeaddrinfo, getaddrinfo — get address information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

void freeaddrinfo(struct addrinfo *ai);
int getaddrinfo(const char *restrict nodename,
   const char *restrict servname,
   const struct addrinfo *restrict hints,
   struct addrinfo **restrict res);
```

DESCRIPTION

The *freeaddrinfo*() function shall free one or more **addrinfo** structures returned by *getaddrinfo*(), along with any additional storage associated with those structures. If the *ai_next* field of the structure is not null, the entire list of structures shall be freed. The *freeaddrinfo*() function shall support the freeing of arbitrary sublists of an **addrinfo** list originally returned by *getaddrinfo*().

The *getaddrinfo*() function shall translate the name of a service location (for example, a host name) and/or a service name and shall return a set of socket addresses and associated information to be used in creating a socket with which to address the specified service.

Note: In many cases it is implemented by the Domain Name System, as documented in RFC 1034, RFC 1035, and RFC 1886.

The *freeaddrinfo*() and *getaddrinfo*() functions shall be thread-safe.

The *nodename* and *servname* arguments are either null pointers or pointers to null-terminated strings. One or both of these two arguments shall be supplied by the application as a non-null pointer.

The format of a valid name depends on the address family or families. If a specific family is not given and the name could be interpreted as valid within multiple supported families, the implementation shall attempt to resolve the name in all supported families and, in absence of errors, one or more results shall be returned.

If the *nodename* argument is not null, it can be a descriptive name or can be an address string. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, valid descriptive names include host names. If the specified address family is AF_INET or AF_UNSPEC, address strings using Internet standard dot notation as specified in *inet_addr*( ) are valid.

If the specified address family is AF_INET6 or AF_UNSPEC, standard IPv6 text forms described in *inet_ntop*( ) are valid.

If *nodename* is not null, the requested service location is named by *nodename*; otherwise, the requested service location is local to the caller.

If *servname* is null, the call shall return network-level addresses for the specified *nodename.* If *servname* is not null, it is a null-terminated character string identifying the requested service. This can be either a descriptive name or a numeric representation suitable for use with the address family or families. If the specified address family is AF_INET, AF_INET6, or AF_UNSPEC, the service can be specified as a string specifying a decimal port number.

If the *hints* argument is not null, it refers to a structure containing input values that directs the operation by providing options and by limiting the returned information to a specific socket type, address family, and/or protocol, as described below. The application shall ensure that each of the *ai_addrlen*, *ai_addr*, *ai_canonname*, and *ai_next* members, as well as each of the non-standard additional members, if any, of this *hints* structure is initialized. If any of these members has a value other than the value that would result from default initialization, the behavior is implementation-defined. A value of AF_UNSPEC for *ai_family* means that the caller shall accept any address family. A value of zero for *ai_socktype* means that the caller shall accept any socket type. A value of zero for *ai_protocol* means that the caller shall accept any protocol. If *hints* is a null pointer, the behavior shall be as if it referred to a structure containing the value zero for the *ai_flags*, *ai_socktype*, and *ai_protocol* fields, and AF_UNSPEC for the *ai_family* field.

The *ai_flags* field to which the *hints* parameter points shall be set to zero or be the bitwise-inclusive OR of one or more of the values AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST, AI_NUMERICSERV, AI_V4MAPPED, AI_ALL, and AI_ADDRCONFIG.

If the AI_PASSIVE flag is specified, the returned address information shall be suitable for use in binding a socket for accepting incoming connections for the specified service. In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to INADDR_ANY for an IPv4 address or IN6ADDR_ANY_INIT for an IPv6 address. If the AI_PASSIVE flag is not specified, the returned address information shall be suitable for a call to *connect*() (for a connection-mode protocol) or for a call to *connect*(), *sendto*(), or *sendmsg*() (for a connectionless protocol). In this case, if the *nodename* argument is null, then the IP address portion of the socket address structure shall be set to the loopback address. The AI_PASSIVE flag shall be ignored if the *nodename* argument is not null.

If the AI_CANONNAME flag is specified and the *nodename* argument is not null, the function shall attempt to determine the canonical name corresponding to *nodename* (for example, if *nodename* is an alias or shorthand notation for a complete name).

Note: Since different implementations use different conceptual models, the terms ''canonical name'' and ''alias'' cannot be precisely defined for the general case. However, Domain Name System implementations are expected to interpret them as they are used in RFC 1034.

A numeric host address string is not a ''name'', and thus does not have a ''canonical name'' form; no address to host name translation is performed. See below for handling of the case where a canonical name cannot be obtained.

If the AI_NUMERICHOST flag is specified, then a non-null *nodename* string supplied shall be a numeric host address string. Otherwise, an **[EAI_NONAME]** error is returned. This flag shall prevent any type of name resolution service (for example, the DNS) from being invoked.

If the AI_NUMERICSERV flag is specified, then a non-null *servname* string supplied shall be a numeric port string. Otherwise, an **[EAI_NONAME]** error shall be returned. This flag shall prevent any type of name resolution service (for example, NIS+) from being invoked.

By default, with an *ai_family* of AF_INET6, *getaddrinfo*() shall return only IPv6 addresses. If the AI_V4MAPPED flag is specified along with an *ai_family* of AF_INET6, then *getaddrinfo*() shall return IPv4-mapped IPv6 addresses on finding no matching IPv6 addresses. The AI_V4MAPPED flag shall be ignored unless *ai_family* equals AF_INET6. If the AI_ALL flag is used with the AI_V4MAPPED flag, then *getaddrinfo*() shall return all matching IPv6 and IPv4 addresses. The AI_ALL flag without the AI_V4MAPPED flag shall be ignored.

If the AI_ADDRCONFIG flag is specified, IPv4 addresses shall be returned only if an IPv4 address is configured on the local system, and IPv6 addresses shall be returned only if an IPv6 address is configured on the local system.

The *ai_socktype* field to which argument *hints* points specifies the socket type for the service, as defined in *socket*( ). If a specific socket type is not given (for example, a value of zero) and the service name could be interpreted as valid with multiple supported socket types, the implementation shall attempt to resolve the service name for all supported socket types and, in the absence of errors, all possible results shall be returned. A non-zero socket type value shall limit the returned information to values with the specified socket type.

If the *ai_family* field to which *hints* points has the value AF_UNSPEC, addresses shall be returned for use with any address family that can be used with the specified *nodename* and/or *servname*. Otherwise, addresses shall be returned for use only with the specified address family. If *ai_family* is not AF_UNSPEC and *ai_protocol* is not zero, then addresses shall be returned for use only with the specified address family and protocol; the value of *ai_protocol* shall be interpreted as in a call to the *socket*() function with the corresponding values of *ai_family* and *ai_protocol*.

RETURN VALUE

A zero return value for *getaddrinfo*() indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful return of *getaddrinfo*(), the location to which *res* points shall refer to a linked list of **addrinfo** structures, each of which shall specify a socket address and information for use in creating a socket with which to use that socket address. The list shall include at least one **addrinfo** structure. The *ai_next* field of each structure contains a pointer to the next structure on the list, or a null pointer if it is the last structure on the list. Each structure on the list shall include values for use with a call to the *socket*() function, and a socket address for use with the *connect*() function or, if the AI_PASSIVE flag was specified, for use with the *bind*() function. The fields *ai_family*, *ai_socktype*, and *ai_protocol* shall be usable as the arguments to the *socket*() function to create a socket suitable for use with the returned address. The fields *ai_addr* and *ai_addrlen* are usable as the arguments to the *connect*() or *bind*() functions with such a socket, according to the AI_PASSIVE flag.

If *nodename* is not null, and if requested by the AI_CANONNAME flag, the *ai_canonname* field of the first returned **addrinfo** structure shall point to a null-terminated string containing the canonical name corresponding to the input *nodename*; if the canonical name is not available, then *ai_canonname* shall refer to the *nodename* argument or a string with the same contents. The contents of the *ai_flags* field of the returned structures are undefined.

All fields in socket address structures returned by *getaddrinfo*() that are not filled in through an explicit argument (for example, *sin6_flowinfo*) shall be set to zero.

Note: This makes it easier to compare socket address structures.

ERRORS

The *getaddrinfo*() function shall fail and return the corresponding error value if:

[EAI_AGAIN]
The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS]
The *flags* parameter had an invalid value.

[EAI_FAIL] A non-recoverable error occurred when attempting to resolve the name.

[EAI_FAMILY]
The address family was not recognized.

[EAI_MEMORY]
There was a memory allocation failure when trying to allocate storage for the return value.

[EAI_NONAME]
The name does not resolve for the supplied parameters.

Neither *nodename* nor *servname* were supplied. At least one of these shall be supplied.

[EAI_SERVICE]
The service passed was not recognized for the specified socket type.

[EAI_SOCKTYPE]
The intended socket type was not recognized.

[EAI_SYSTEM]
A system error occurred; the error code can be found in *errno*.

The following sections are informative.

EXAMPLES

The following (incomplete) program demonstrates the use of *getaddrinfo*() to obtain the socket address structure(s) for the service named in the program's command-line argument. The program then loops through each of the address structures attempting to create and bind a socket to the address, until it performs a successful *bind*().

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

int
main(int argc, char *argv[])
{
    struct addrinfo *result, *rp;
    int sfd, s;

    if (argc != 2) {
        fprintf(stderr, "Usage: %s port\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    struct addrinfo hints = {0};
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_DGRAM;
    hints.ai_flags = AI_PASSIVE;
    hints.ai_protocol = 0;

    s = getaddrinfo(NULL, argv[1], &hints, &result);
    if (s != 0) {
        fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(s));
        exit(EXIT_FAILURE);
    }

    /* getaddrinfo() returns a list of address structures.
       Try each address until a successful bind().
       If socket(2) (or bind(2)) fails, close the socket
       and try the next address. */

    for (rp = result; rp != NULL; rp = rp->ai_next) {
        sfd = socket(rp->ai_family, rp->ai_socktype,
            rp->ai_protocol);
        if (sfd == -1)
            continue;

        if (bind(sfd, rp->ai_addr, rp->ai_addrlen) == 0)
            break;           /* Success */

        close(sfd);
    }

    if (rp == NULL) {        /* No address succeeded */
        fprintf(stderr, "Could not bind\n");
        exit(EXIT_FAILURE);
    }

    freeaddrinfo(result);    /* No longer needed */

        /* ... use socket bound to sfd ... */
}
```

APPLICATION USAGE

If the caller handles only TCP and not UDP, for example, then the *ai_protocol* member of the *hints* structure should be set to IPPROTO_TCP when *getaddrinfo*() is called.

If the caller handles only IPv4 and not IPv6, then the *ai_family* member of the *hints* structure should be set to AF_INET when *getaddrinfo*() is called.

Although it is common practice to initialize the *hints* structure using:

```
struct addrinfo hints;
memset(&hints, 0, sizeof hints);
```

this method is not portable according to this standard, because the structure can contain pointer or floating-point members that are not required to have an all-bits-zero representation after default initialization. Portable methods make use of default initialization; for example:

```
struct addrinfo hints = { 0 };
```

or:

```
static struct addrinfo hints_init;
struct addrinfo hints = hints_init;
```

A future version of this standard may require that a pointer object with an all-bits-zero representation is a null pointer, and that **addrinfo** does not have any floating-point members if a floating-point object with an all-bits-zero representation does not have the value 0.0.

The term "canonical name" is misleading; it is taken from the Domain Name System (RFC 2181). It should be noted that the canonical name is a result of alias processing, and not necessarily a unique attribute of a host, address, or set of addresses. See RFC 2181 for more discussion of this in the Domain Name System context.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *endservent*( ), *gai_strerror*( ), *getnameinfo*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

freelocale — free resources allocated for a locale object

SYNOPSIS

#include <locale.h>

void freelocale(locale_t *locobj*);

DESCRIPTION

The *freelocale*() function shall cause the resources allocated for a locale object returned by a call to the *newlocale*() or *duplocale*() functions to be released.

The behavior is undefined if the *locobj* argument is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

Any use of a locale object that has been freed results in undefined behavior.

RETURN VALUE

None.

ERRORS

None.

The following sections are informative.

EXAMPLES

Freeing Up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale*():

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", NULL);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *newlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

freopen — open a stream

SYNOPSIS

```
#include <stdio.h>

FILE *freopen(const char *restrict pathname, const char *restrict mode,
   FILE *restrict stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *freopen*() function shall first attempt to flush the stream associated with *stream* as if by a call to *fflush*(*stream*). Failure to flush the stream successfully shall be ignored. If *pathname* is not a null pointer, *freopen*() shall close any file descriptor associated with *stream*. Failure to close the file descriptor successfully shall be ignored. The error and end-of-file indicators for the stream shall be cleared.

The *freopen*() function shall open the file whose pathname is the string pointed to by *pathname* and associate the stream pointed to by *stream* with it. The *mode* argument shall be used just as in *fopen*().

The original stream shall be closed regardless of whether the subsequent open succeeds.

If *pathname* is a null pointer, the *freopen*() function shall attempt to change the mode of the stream to that specified by *mode*, as if the name of the file currently associated with the stream had been used. In this case, the file descriptor associated with the stream need not be closed if the call to *freopen*() succeeds. It is implementation-defined which changes of mode are permitted (if any), and under what circumstances.

After a successful call to the *freopen*() function, the orientation of the stream shall be cleared, the encoding rule shall be cleared, and the associated **mbstate_t** object shall be set to describe an initial conversion state.

If *pathname* is not a null pointer, or if *pathname* is a null pointer and the specified mode change necessitates the file descriptor associated with the stream to be closed and reopened, the file descriptor associated with the reopened stream shall be allocated and opened as if by a call to *open*() with the following flags:

center box tab(!); cB | cB l | l. *freopen*() Mode!*open*() Flags _ *r* or *rb*!O_RDONLY *w* or *wb*!O_WRONLY|O_CREAT|O_TRUNC *a* or *ab*!O_WRONLY|O_CREAT|O_APPEND *r*+ or *rb*+ or *r*+*b*!O_RDWR *w*+ or *wb*+ or *w*+*b*!O_RDWR|O_CREAT|O_TRUNC *a*+ or *ab*+ or *a*+*b*!O_RDWR|O_CREAT|O_APPEND

RETURN VALUE

Upon successful completion, *freopen*() shall return the value of *stream*. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

The *freopen*() function shall fail if:

EACCES

Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *mode* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created.

EBADF

The file descriptor underlying the stream is not a valid file descriptor when *pathname* is a null pointer.

EINTR
A signal was caught during *freopen*().

EISDIR
The named file is a directory and *mode* requires write access.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENFILE
The maximum allowable number of files is currently open in the system.

ENOENT
The *mode* string begins with **'r'** and a component of *pathname* does not name an existing file, or *mode* begins with **'w'** or **'a'** and a component of the path prefix of *pathname* does not name an existing file, or *pathname* is an empty string.

ENOENT or **ENOTDIR**
The *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *pathname* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory or file system that would contain the new file cannot be expanded, the file does not exist, and it was to be created.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *pathname* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

ENXIO
The named file is a character special or block special file, and the device associated with this special file does not exist.

EOVERFLOW
The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

EROFS
The named file resides on a read-only file system and *mode* requires write access.

The *freopen*() function may fail if:

EBADF
The mode with which the file descriptor underlying the stream was opened does not support the requested mode when *pathname* is a null pointer.

EINVAL
The value of the *mode* argument is not valid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENOMEM
Insufficient storage space is available.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

ETXTBSY
The file is a pure procedure (shared text) file that is being executed and *mode* requires write access.

The following sections are informative.

EXAMPLES

Directing Standard Output to a File

The following example logs all standard output to the **/tmp/logfile** file.

```
#include <stdio.h>
...
FILE *fp;
...
fp = freopen ("/tmp/logfile", "a+", stdout);
...
```

APPLICATION USAGE

The *freopen*() function is typically used to attach the pre-opened *streams* associated with *stdin*, *stdout*, and *stderr* to other files.

Since implementations are not required to support any stream mode changes when the *pathname* argument is NULL, portable applications cannot rely on the use of *freopen*() to change the stream mode, and use of this feature is discouraged. The feature was originally added to the ISO C standard in order to facilitate changing *stdin* and *stdout* to binary mode. Since a **'b'** character in the mode has no effect on POSIX systems, this use of the feature is unnecessary in POSIX applications. However, even though the **'b'** is ignored, a successful call to *freopen*(NULL, "*wb*", *stdout*) does have an effect. In particular, for regular files it truncates the file and sets the file-position indicator for the stream to the start of the file. It is possible that these side-effects are an unintended consequence of the way the feature is specified in the ISO/IEC 9899: 1999 standard, but unless or until the ISO C standard is changed, applications which successfully call *freopen*(NULL, "*wb*", *stdout*) will behave in unexpected ways on conforming systems in situations such as:

```
{ appl file1; appl file2; } > file3
```

which will result in **file3** containing only the output from the second invocation of *appl*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fclose*( ), *fdopen*( ), *fflush*( ), *fmemopen*( ), *fopen*( ), *mbsinit*( ), *open*( ), *open_memstream*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

frexp, frexpf, frexpl — extract mantissa and exponent from a double precision number

SYNOPSIS

#include <math.h>

double frexp(double *num*, int **exp*);
float frexpf(float *num*, int **exp*);
long double frexpl(long double *num*, int **exp*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall break a floating-point number *num* into a normalized fraction and an integral power of 2. The integer exponent shall be stored in the **int** object pointed to by *exp*.

RETURN VALUE

For finite arguments, these functions shall return the value *x*, such that *x* has a magnitude in the interval [½,1) or 0, and *num* equals *x* times 2 raised to the power **exp*.

If *num* is NaN, a NaN shall be returned, and the value of **exp* is unspecified.

If *num* is ±0, ±0 shall be returned, and the value of **exp* shall be 0.

If *num* is ±Inf, *num* shall be returned, and the value of **exp* is unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isnan( ), *ldexp*( ), *modf*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fscanf, scanf, sscanf — convert formatted input

SYNOPSIS

#include <stdio.h>

int fscanf(FILE *restrict *stream*, const char *restrict *format*, ...);
int scanf(const char *restrict *format*, ...);
int sscanf(const char *restrict *s*, const char *restrict *format*, ...);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fscanf*() function shall read from the named input *stream*. The *scanf*() function shall read from the standard input stream *stdin*. The *sscanf*() function shall read from the string *s*. Each function reads bytes, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments shall be evaluated but otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier character **%** (see below) is replaced by the sequence "%*n*$", where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of format strings that select arguments in an order appropriate to specific languages. In format strings containing the "%*n*$" form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the format string more than once.

The *format* can contain either form of a conversion specification—that is, **%** or "%*n*$"—but the two forms cannot be mixed within a single *format* string. The only exception to this is that **%%** or **%*** can be mixed with the "%*n*$" form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The *fscanf*() function in all its forms shall allow detection of a language-dependent radix character in the input string. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (**'.'**).

The format is a character string, beginning and ending in its initial shift state, if any, composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary character (neither **'%'** nor a white-space character); or a conversion specification. Each conversion specification is introduced by the character **'%'** or the character sequence "%*n*$", after which the following appear in sequence:

* An optional assignment-suppressing character **'*'**.
* An optional non-zero decimal integer that specifies the maximum field width.
* An optional assignment-allocation character **'m'**.
* An option length modifier that specifies the size of the receiving object.
* A *conversion specifier* character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fscanf*() functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space characters shall be executed by reading input until no more valid input can be read, or up to the first byte which is not a white-space character, which remains unread.

A directive that is an ordinary character shall be executed as follows: the next byte shall be read from the input and compared with the byte that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent bytes shall remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a character from being read, the directive shall fail.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion character. A conversion specification shall be executed in the following steps.

Input white-space characters (as specified by *isspace*( )) shall be skipped, unless the conversion specification includes a **[**, **c**, **C**, or **n** conversion specifier.

An item shall be read from the input, unless the conversion specification includes an **n** conversion specifier. An input item shall be defined as the longest sequence of input bytes (up to any specified maximum field width, which may be measured in characters or bytes dependent on the conversion specifier) which is an initial subsequence of a matching sequence. The first byte, if any, after the input item shall remain unread. If the length of the input item is 0, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a **%** conversion specifier, the input item (or, in the case of a **%n** conversion specification, the count of input bytes) shall be converted to a type appropriate to the conversion character. If the input item is not a matching sequence, the execution of the conversion specification fails; this condition is a matching failure. Unless assignment suppression was indicated by a **'*'**, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by **%**, or in the *n*th argument if introduced by the character sequence "%*n*$". If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The **%c**, **%s**, and **%[** conversion specifiers shall accept an optional assignment-allocation character **'m'**, which shall cause a memory buffer to be allocated to hold the string converted including a terminating null character. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer variable that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc*() had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to **[ENOMEM]** and a conversion error shall result. If the function returns EOF, any memory successfully allocated for parameters using assignment-allocation character **'m'** by this call shall be freed before the function returns.

The length modifiers and their meanings are:

hh Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **signed char** or **unsigned char**.

h Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **short** or **unsigned short**.

l (ell) Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **long** or **unsigned long**; that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to an argument with type pointer to **double**; or that a following **c**, **s**, or **[** conversion specifier applies to an argument with type pointer to **wchar_t**. If the **'m'** assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to **wchar_t**.

ll (ell-ell)

Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **long long** or **unsigned long long**.

j Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **intmax_t** or **uintmax_t**.

z Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **size_t** or the corresponding signed integer type.

t Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

L Specifies that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to an argument with type pointer to **long double**.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The following conversion specifiers are valid:

d Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *strtol*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

i Matches an optionally signed integer, whose format is the same as expected for the subject sequence of *strtol*() with 0 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

o Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 8 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

u Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

x Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of *strtoul*() with the value 16 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

a, e, f, g

Matches an optionally signed floating-point number, infinity, or NaN, whose format is the same as expected for the subject sequence of *strtod*(). In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **float**.

If the *fprintf*() family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the *fscanf*() family of functions shall recognize them as input.

s Matches a sequence of bytes that are not white-space characters. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence and a terminating null character code, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If an **l** (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character shall be converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

[Matches a non-empty sequence of bytes from a set of expected bytes (the *scanset*). The normal skip over white-space characters shall be suppressed in this case. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence and a terminating null byte, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If an **l** (ell) qualifier is present, the input is a sequence of characters that begins in the initial shift state. Each character in the sequence shall be converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which shall be added automatically.
Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

The conversion specification includes all subsequent bytes in the *format* string up to and including the matching <right-square-bracket> (**']'**). The bytes between the square brackets (the *scanlist*) comprise the scanset, unless the byte after the <left-square-bracket> is a <circumflex> (**'^'**), in which case the scanset contains all bytes that do not appear in the scanlist between the <circumflex> and the <right-square-bracket>. If the conversion specification begins with **"[ ]"** or **"[^]"**, the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a **'−'** is in the scanlist and is not the first character, nor the second where the first character is a **'^'**, nor the last character, the behavior is implementation-defined.

c Matches a sequence of bytes of the number specified by the field width (1 if no field width is present in the conversion specification). No null byte is added. The normal skip over white-space characters shall be suppressed in this case. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial byte of an array of **char**, **signed char**, or **unsigned char** large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

If an **l** (ell) qualifier is present, the input shall be a sequence of characters that begins in the initial shift state. Each character in the sequence is converted to a wide character as if by a call to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted. No null wide character is added. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the resulting sequence of wide characters. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

p Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the **%p** conversion specification of the corresponding *fprintf*() functions. The application shall ensure that the corresponding argument is a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the **%p** conversion specification is undefined.

n No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which shall be written the number of bytes read from the input so far by this call to the *fscanf*() functions. Execution of a **%n** conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing character or a field width, the behavior is undefined.

C Equivalent to **lc**.

S Equivalent to **ls**.

% Matches a single **'%'** character; no conversion or assignment occurs. The complete conversion specification shall be **%%**.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **A**, **E**, **F**, **G**, and **X** are also valid and shall be equivalent to **a**, **e**, **f**, **g**, and **x**, respectively.

If end-of-file is encountered during input, conversion shall be terminated. If end-of-file occurs before any bytes matching the current conversion specification (except for **%n**) have been read (other than leading white-space characters, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *sscanf*() shall be equivalent to encountering end-of-file for *fscanf*().

If conversion terminates on a conflicting input, the offending input is left unread in the input. Any trailing white space (including <newline> characters) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the **%n** conversion specification.

The *fscanf*() and *scanf*() functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), *fscanf*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned and *errno* shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the *fscanf*() functions fail and may fail, refer to *fgetc*( ) or *fgetwc*( ).

In addition, the *fscanf*() function shall fail if:

EILSEQ
Input byte sequence does not form a valid character.

ENOMEM
Insufficient storage space is available.

In addition, the *fscanf*() function may fail if:

EINVAL
There are insufficient arguments.

The following sections are informative.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = scanf("%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string **"Hamster"**.

The call:

```
int i; float x; char name[50];
(void) scanf("%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string **"56\0"** in *name*. The next call to *getchar*() shall return the character **'a'**.

Reading Data into an Array

The following call uses *fscanf*() to read three floating-point numbers from standard input into the *input* array.

```
float input[3]; fscanf (stdin, "%f %f %f", input, input+1, input+2);
```

APPLICATION USAGE

If the application calling *fscanf*() has any objects of type **wint_t** or **wchar_t**, it must also include the *<wchar.h>* header to have these objects defined.

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *fscanf*(), this is memory allocated via use of the **'m'** assignment-allocation character.

RATIONALE

This function is aligned with the ISO/IEC 9899: 1999 standard, and in doing so a few "obvious" things were not included. Specifically, the set of characters allowed in a scanset is limited to single-byte characters. In other similar places, multi-byte characters have been permitted, but for alignment with the ISO/IEC 9899: 1999 standard, it has not been done here. Applications needing this could use the corresponding wide-character functions to achieve the desired results.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fprintf*( ), *getc*( ), *setlocale*( ), *strtod*( ), *strtol*( ), *strtoul*( ), *wcrtomb*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<inttypes.h>**, **<langinfo.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fseek, fseeko — reposition a file-position indicator in a stream

SYNOPSIS

#include <stdio.h>

int fseek(FILE **stream*, long *offset*, int *whence*);
int fseeko(FILE **stream*, off_t *offset*, int *whence*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fseek*() function shall set the file-position indicator for the stream pointed to by *stream*. If a read or write error occurs, the error indicator for the stream shall be set and *fseek*() fails.

The new position, measured in bytes from the beginning of the file, shall be obtained by adding *offset* to the position specified by *whence*. The specified point is the beginning of the file for SEEK_SET, the current value of the file-position indicator for SEEK_CUR, or end-of-file for SEEK_END.

If the stream is to be used with wide-character input/output functions, the application shall ensure that *offset* is either 0 or a value returned by an earlier call to *ftell*() on the same stream and *whence* is SEEK_SET.

A successful call to *fseek*() shall clear the end-of-file indicator for the stream and undo any effects of *ungetc*() and *ungetwc*() on the same stream. After an *fseek*() call, the next operation on an update stream may be either input or output.

If the most recent operation, other than *ftell*(), on a given stream is *fflush*(), the file offset in the underlying open file description shall be adjusted to reflect the location specified by *fseek*().

The *fseek*() function shall allow the file-position indicator to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The behavior of *fseek*() on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

If the stream is writable and buffered data had not been written to the underlying file, *fseek*() shall cause the unwritten data to be written to the file and shall mark the last data modification and last file status change timestamps of the file for update.

In a locale with state-dependent encoding, whether *fseek*() restores the stream's shift state is implementation-defined.

The *fseeko*() function shall be equivalent to the *fseek*() function except that the *offset* argument is of type **off_t**.

RETURN VALUE

The *fseek*() and *fseeko*() functions shall return 0 if they succeed.

Otherwise, they shall return −1 and set *errno* to indicate the error.

ERRORS

The *fseek*() and *fseeko*() functions shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to be flushed, and the call to *fseek*() or *fseeko*() causes an underlying *lseek*() or *write*() to be invoked, and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.

EFBIG
An attempt was made to write a file that exceeds the maximum file size.

EFBIG
An attempt was made to write a file that exceeds the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EINVAL
The *whence* argument is invalid. The resulting file-position indicator would be set to a negative value.

EIO A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a *write*() to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOSPC
There was no free space remaining on the device containing the file.

EOVERFLOW
For *fseek*(), the resulting file offset would be a value which cannot be represented correctly in an object of type **long**.

EOVERFLOW
For *fseeko*(), the resulting file offset would be a value which cannot be represented correctly in an object of type **off_t**.

EPIPE An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.

ESPIPE
The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

The *fseek*() and *fseeko*() functions may fail if:

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *fsetpos*( ), *ftell*( ), *getrlimit*( ), *lseek*( ), *rewind*( ), *ulimit*( ), *ungetc*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fsetpos — set current file position

SYNOPSIS

#include <stdio.h>

int fsetpos(FILE **stream*, const fpos_t **pos*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fsetpos*() function shall set the file position and state indicators for the stream pointed to by *stream* according to the value of the object pointed to by *pos*, which the application shall ensure is a value obtained from an earlier call to *fgetpos*() on the same stream. If a read or write error occurs, the error indicator for the stream shall be set and *fsetpos*() fails.

A successful call to the *fsetpos*() function shall clear the end-of-file indicator for the stream and undo any effects of *ungetc*() on the same stream. After an *fsetpos*() call, the next operation on an update stream may be either input or output.

The behavior of *fsetpos*() on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The *fsetpos*() function shall not change the setting of *errno* if successful.

RETURN VALUE

The *fsetpos*() function shall return 0 if it succeeds; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *fsetpos*() function shall fail if, either the *stream* is unbuffered or the *stream*'s buffer needed to be flushed, and the call to *fsetpos*() causes an underlying *lseek*() or *write*() to be invoked, and:

EAGAIN
The O_NONBLOCK flag is set for the file descriptor and the thread would be delayed in the write operation.

EBADF
The file descriptor underlying the stream file is not open for writing or the stream's buffer needed to be flushed and the file is not open.

EFBIG
An attempt was made to write a file that exceeds the maximum file size.

EFBIG
An attempt was made to write a file that exceeds the file size limit of the process.

EFBIG
The file is a regular file and an attempt was made to write at or beyond the offset maximum associated with the corresponding stream.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EIO A physical I/O error has occurred, or the process is a member of a background process group attempting to perform a *write*() to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOSPC
There was no free space remaining on the device containing the file.

EPIPE An attempt was made to write to a pipe or FIFO that is not open for reading by any process; a SIGPIPE signal shall also be sent to the thread.

ESPIPE
The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

The *fsetpos*() function may fail if:

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *ftell*( ), *lseek*( ), *rewind*( ), *ungetc*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fstat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int fstat(int fildes, struct stat *buf);
```

DESCRIPTION

The *fstat*() function shall obtain information about an open file associated with the file descriptor *fildes*, and shall write it to the area pointed to by *buf*.

If *fildes* references a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

If *fildes* references a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

The *buf* argument is a pointer to a **stat** structure, as defined in *<sys/stat.h>*, into which information is placed concerning the file.

For all other file types defined in this volume of POSIX.1-2017, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the *st_nlink* member shall be set to the number of links to the file.

An implementation that provides additional or alternative file access control mechanisms may, under implementation-defined conditions, cause *fstat*() to fail.

The *fstat*() function shall update any time-related fields (as described in the Base Definitions volume of POSIX.1-2017, *Section 4.9*, *File Times Update*), before writing into the **stat** structure.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *fstat*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EIO An I/O error occurred while reading from the file system.

EOVERFLOW
The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

The *fstat*() function may fail if:

EOVERFLOW
One of the values is too large to store into the structure pointed to by the *buf* argument.

The following sections are informative.

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure. The **/home/cnd/mod1** file is opened with read/write privileges and is passed to the open file descriptor *fildes*.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = fstat(fildes, &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fstatat( )

The Base Definitions volume of POSIX.1-2017, *Section 4.9*, *File Times Update*, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fstatat, lstat, stat — get file status

SYNOPSIS

```
#include <fcntl.h>
#include <sys/stat.h>

int fstatat(int fd, const char *restrict path,
    struct stat *restrict buf, int flag);
int lstat(const char *restrict path, struct stat *restrict buf);
int stat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

The *stat*() function shall obtain information about the named file and write it to the area pointed to by the *buf* argument. The *path* argument points to a pathname naming a file. Read, write, or execute permission of the named file is not required. An implementation that provides additional or alternate file access control mechanisms may, under implementation-defined conditions, cause *stat*() to fail. In particular, the system may deny the existence of the file specified by *path*.

If the named file is a symbolic link, the *stat*() function shall continue pathname resolution using the contents of the symbolic link, and shall return information pertaining to the resulting file if the file exists.

The *buf* argument is a pointer to a **stat** structure, as defined in the *<sys/stat.h>* header, into which information is placed concerning the file.

The *stat*() function shall update any time-related fields (as described in the Base Definitions volume of POSIX.1-2017, *Section 4.9*, *File Times Update*), before writing into the **stat** structure.

If the named file is a shared memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

If the named file is a typed memory object, the implementation shall update in the **stat** structure pointed to by the *buf* argument the *st_uid*, *st_gid*, *st_size*, and *st_mode* fields, and only the S_IRUSR, S_IWUSR, S_IRGRP, S_IWGRP, S_IROTH, and S_IWOTH file permission bits need be valid. The implementation may update other fields and flags.

For all other file types defined in this volume of POSIX.1-2017, the structure members *st_mode*, *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the member *st_nlink* shall be set to the number of links to the file.

The *lstat*() function shall be equivalent to *stat*(), except when *path* refers to a symbolic link. In that case *lstat*() shall return information about the link, while *stat*() shall return information about the file the link references.

For symbolic links, the *st_mode* member shall contain meaningful information when used with the file type macros. The file mode bits in *st_mode* are unspecified. The structure members *st_ino*, *st_dev*, *st_uid*, *st_gid*, *st_atim*, *st_ctim*, and *st_mtim* shall have meaningful values and the value of the *st_nlink* member shall be set to the number of (hard) links to the symbolic link. The value of the *st_size* member shall be set to the length of the pathname contained in the symbolic link not including any terminating null byte.

The *fstatat*() function shall be equivalent to the *stat*() or *lstat*() function, depending on the value of *flag* (see below), except in the case where *path* specifies a relative path. In this case the status shall be retrieved from a file relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in <*fcntl.h*>:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, the status of the symbolic link is returned.

If *fstatat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *stat*() or *lstat*() respectively, depending on whether or not the AT_SYMLINK_NOFOLLOW bit is set in *flag*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EACCES
Search permission is denied for a component of the path prefix.

EIO An error occurred while reading from the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EOVERFLOW
The file size in bytes or the number of blocks allocated to the file or the file serial number cannot be represented correctly in the structure pointed to by *buf*.

The *fstatat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

EOVERFLOW
A value to be stored would overflow one of the members of the **stat** structure.

The *fstatat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

Obtaining File Status Information

The following example shows how to obtain file status information for a file named **/home/cnd/mod1**. The structure variable *buffer* is defined for the **stat** structure.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

struct stat buffer;
int         status;
...
status = stat("/home/cnd/mod1", &buffer);
```

Getting Directory Information

The following example fragment gets status information for each entry in a directory. The call to the *stat*() function stores file information in the **stat** structure pointed to by *statbuf*. The lines that follow the *stat*() call format the fields in the **stat** structure for presentation to the user of the program.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <dirent.h>
#include <pwd.h>
#include <grp.h>
#include <time.h>
#include <locale.h>
#include <langinfo.h>
#include <stdio.h>
#include <stdint.h>

struct dirent  *dp;
struct stat     statbuf;
struct passwd  *pwd;
struct group   *grp;
struct tm      *tm;
char            datestring[256];
...
/* Loop through directory entries. */
while ((dp = readdir(dir)) != NULL) {

    /* Get entry's information. */
    if (stat(dp->d_name, &statbuf) == -1)
        continue;

    /* Print out type, permissions, and number of links. */
    printf("%10.10s", sperm (statbuf.st_mode));
    printf("%4d", statbuf.st_nlink);
```

```
        /* Print out owner's name if it is found using getpwuid(). */
        if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
            printf(" %-8.8s", pwd->pw_name);
        else
            printf(" %-8d", statbuf.st_uid);

        /* Print out group name if it is found using getgrgid(). */
        if ((grp = getgrgid(statbuf.st_gid)) != NULL)
            printf(" %-8.8s", grp->gr_name);
        else
            printf(" %-8d", statbuf.st_gid);

        /* Print size of file. */
        printf(" %9jd", (intmax_t)statbuf.st_size);

        tm = localtime(&statbuf.st_mtime);

        /* Get localized date string. */
        strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);

        printf(" %s %s\n", datestring, dp->d_name);
    }
```

Obtaining Symbolic Link Status Information

The following example shows how to obtain status information for a symbolic link named **/modules/pass1**. The structure variable *buffer* is defined for the **stat** structure. If the *path* argument specified the pathname for the file pointed to by the symbolic link (**/home/cnd/mod1**), the results of calling the function would be the same as those returned by a call to the *stat*() function.

```
#include <sys/stat.h>

struct stat buffer;
int status;
...
status = lstat("/modules/pass1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

The intent of the paragraph describing ‘‘additional or alternate file access control mechanisms’’ is to allow a secure implementation where a process with a label that does not dominate the file's label cannot perform a *stat*() function. This is not related to read permission; a process with a label that dominates the file's label does not need read permission. An implementation that supports write-up operations could fail *fstat*() function calls even though it has a valid file descriptor open for writing.

The purpose of the *fstatat*() function is to obtain the status of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *stat*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *fstatat*() function it can be guaranteed that the file for which status is returned is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

access( ), *chmod*( ), *fdopendir*( ), *fstat*( ), *mknod*( ), *readlink*( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.9*, *File Times Update*, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fstatvfs, statvfs — get file system information

SYNOPSIS

```
#include <sys/statvfs.h>

int fstatvfs(int fildes, struct statvfs *buf);
int statvfs(const char *restrict path, struct statvfs *restrict buf);
```

DESCRIPTION

The *fstatvfs*() function shall obtain information about the file system containing the file referenced by *fildes*.

The *statvfs*() function shall obtain information about the file system containing the file named by *path*.

For both functions, the *buf* argument is a pointer to a **statvfs** structure that shall be filled. Read, write, or execute permission of the named file is not required.

The following flags can be returned in the *f_flag* member:

ST_RDONLY	Read-only file system.
ST_NOSUID	Setuid/setgid bits ignored by *exec*.

It is unspecified whether all members of the **statvfs** structure have meaningful values on all file systems.

RETURN VALUE

Upon successful completion, *statvfs*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *fstatvfs*() and *statvfs*() functions shall fail if:

EIO An I/O error occurred while reading the file system.

EINTR
A signal was caught during execution of the function.

EOVERFLOW
One of the values to be returned cannot be represented correctly in the structure pointed to by *buf*.

The *fstatvfs*() function shall fail if:

EBADF
The *fildes* argument is not an open file descriptor.

The *statvfs*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *statvfs*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Obtaining File System Information Using fstatvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *fstatvfs*() function. The **/home/cnd/mod1** file is opened with read/write privileges and the open file descriptor is passed to the *fstatvfs*() function.

```
#include <sys/statvfs.h>
#include <fcntl.h>

struct statvfs buffer;
int          status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status  = fstatvfs(fildes, &buffer);
```

Obtaining File System Information Using statvfs( )

The following example shows how to obtain file system information for the file system upon which the file named **/home/cnd/mod1** resides, using the *statvfs*() function.

```
#include <sys/statvfs.h>

struct statvfs buffer;
int          status;
...
status = statvfs("/home/cnd/mod1", &buffer);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *chown*( ), *creat*( ), *dup*( ), *exec*, *fcntl*( ), *link*( ), *mknod*( ), *open*( ), *pipe*( ), *read*( ), *time*( ), *unlink*( ), *utime*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_statvfs.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fsync — synchronize changes to a file

SYNOPSIS

#include <unistd.h>

int fsync(int *fildes*);

DESCRIPTION

The *fsync*() function shall request that all data for the open file descriptor named by *fildes* is to be transferred to the storage device associated with the file described by *fildes*. The nature of the transfer is implementation-defined. The *fsync*() function shall not return until the system has completed that action or until an error is detected.

If _POSIX_SYNCHRONIZED_IO is defined, the *fsync*() function shall force all currently queued I/O operations associated with the file indicated by file descriptor *fildes* to the synchronized I/O completion state. All I/O operations shall be completed as defined for synchronized I/O file integrity completion.

RETURN VALUE

Upon successful completion, *fsync*() shall return 0. Otherwise, −1 shall be returned and *errno* set to indicate the error. If the *fsync*() function fails, outstanding I/O operations are not guaranteed to have been completed.

ERRORS

The *fsync*() function shall fail if:

EBADF
The *fildes* argument is not a valid descriptor.

EINTR
The *fsync*() function was interrupted by a signal.

EINVAL
The *fildes* argument does not refer to a file on which this operation is possible.

EIO An I/O error occurred while reading from or writing to the file system.

In the event that any of the queued I/O operations fail, *fsync*() shall return the error conditions defined for *read*() and *write*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *fsync*() function should be used by programs which require modifications to a file to be completed before continuing; for example, a program which contains a simple transaction facility might use it to ensure that all modifications to a file or files caused by a transaction are recorded.

RATIONALE

The *fsync*() function is intended to force a physical write of data from the buffer cache, and to assure that after a system crash or other failure that all data up to the time of the *fsync*() call is recorded on the disk. Since the concepts of ``buffer cache'', ``system crash'', ``physical write'', and ``non-volatile storage'' are not defined here, the wording has to be more abstract.

If _POSIX_SYNCHRONIZED_IO is not defined, the wording relies heavily on the conformance document to tell the user what can be expected from the system. It is explicitly intended that a null implementation is permitted. This could be valid in the case where the system cannot assure non-volatile storage under any circumstances or when the system is highly fault-tolerant and the functionality is not required. In the middle ground between these extremes, *fsync*() might or might not actually cause data to be written where it is safe from a power failure. The conformance document should identify at least that one configuration exists (and how to obtain that configuration) where this can be assured for at least some files that the user can select to use for critical data. It is not intended that an exhaustive list is required, but rather sufficient information is provided so that if critical data needs to be saved, the user can determine how the system is to be configured to allow the data to be written to non-volatile storage.

It is reasonable to assert that the key aspects of *fsync*() are unreasonable to test in a test suite. That does not make the function any less valuable, just more difficult to test. A formal conformance test should probably force a system crash (power shutdown) during the test for this condition, but it needs to be done in such a way that automated testing does not require this to be done except when a formal record of the results is being made. It would also not be unreasonable to omit testing for *fsync*(), allowing it to be treated as a quality-of-implementation issue.

FUTURE DIRECTIONS

None.

SEE ALSO

sync( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ftell, ftello — return a file offset in a stream

SYNOPSIS

```
#include <stdio.h>

long ftell(FILE *stream);
off_t ftello(FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ftell*() function shall obtain the current value of the file-position indicator for the stream pointed to by *stream*.

The *ftell*() function shall not change the setting of *errno* if successful.

The *ftello*() function shall be equivalent to *ftell*(), except that the return value is of type **off_t** and the *ftello*() function may change the setting of *errno* if successful.

RETURN VALUE

Upon successful completion, *ftell*() and *ftello*() shall return the current value of the file-position indicator for the stream measured in bytes from the beginning of the file.

Otherwise, *ftell*() and *ftello*() shall return −1, and set *errno* to indicate the error.

ERRORS

The *ftell*() and *ftello*() functions shall fail if:

EBADF
The file descriptor underlying *stream* is not an open file descriptor.

EOVERFLOW
For *ftell*(), the current file offset cannot be represented correctly in an object of type **long**.

EOVERFLOW
For *ftello*(), the current file offset cannot be represented correctly in an object of type **off_t**.

ESPIPE
The file descriptor underlying *stream* is associated with a pipe, FIFO, or socket.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetpos*( ), *fopen*( ), *fseek*( ), *lseek*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ftok — generate an IPC key

SYNOPSIS

```
#include <sys/ipc.h>

key_t ftok(const char *path, int id);
```

DESCRIPTION

The *ftok*() function shall return a key based on *path* and *id* that is usable in subsequent calls to *msgget*(), *semget*(), and *shmget*(). The application shall ensure that the *path* argument is the pathname of an existing file that the process is able to *stat*(), with the exception that if *stat*() would fail with **[EOVERFLOW]** due to file size, *ftok*() shall still succeed.

The *ftok*() function shall return the same key value for all paths that name the same file, when called with the same *id* value, and should return different key values when called with different *id* values or with paths that name different files existing on the same file system at the same time. It is unspecified whether *ftok*() shall return the same key value when called again after the file named by *path* is removed and recreated with the same name.

Only the low-order 8-bits of *id* are significant. The behavior of *ftok*() is unspecified if these bits are 0.

RETURN VALUE

Upon successful completion, *ftok*() shall return a key. Otherwise, *ftok*() shall return (**key_t**)−1 and set *errno* to indicate the error.

ERRORS

The *ftok*() function shall fail if:

EACCES
Search permission is denied for a component of the path prefix.

EIO An error occurred while reading from the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *ftok*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Getting an IPC Key

The following example gets a key based on the pathname **/tmp** and the ID value *a*. It also assigns the value of the resulting key to the *semkey* variable so that it will be available to a later call to *semget*(), *msgget*(), or *shmget*().

```
#include <sys/ipc.h>
...
key_t semkey;

if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok"); exit(1);
}
```

APPLICATION USAGE

For maximum portability, *id* should be a single-byte character.

Applications should not assume that the resulting key value is unique.

RATIONALE

None.

FUTURE DIRECTIONS

Future versions of this standard may add new interfaces to provide unique keys.

SEE ALSO

msgget( ), *semget*( ), *shmget*( )

The Base Definitions volume of POSIX.1-2017, **<sys_ipc.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ftruncate — truncate a file to a specified length

SYNOPSIS

#include <unistd.h>

int ftruncate(int *fildes*, off_t *length*);

DESCRIPTION

If *fildes* is not a valid file descriptor open for writing, the *ftruncate*() function shall fail.

If *fildes* refers to a regular file, the *ftruncate*() function shall cause the size of the file to be truncated to *length*. If the size of the file previously exceeded *length*, the extra data shall no longer be available to reads on the file. If the file previously was smaller than this size, *ftruncate*() shall increase the size of the file. If the file size is increased, the extended area shall appear as if it were zero-filled. The value of the seek pointer shall not be modified by a call to *ftruncate*().

Upon successful completion, if *fildes* refers to a regular file, *ftruncate*() shall mark for update the last data modification and last file status change timestamps of the file and the S_ISUID and S_ISGID bits of the file mode may be cleared. If the *ftruncate*() function is unsuccessful, the file is unaffected.

If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

If *fildes* refers to a directory, *ftruncate*() shall fail.

If *fildes* refers to any other file type, except a shared memory object, the result is unspecified.

If *fildes* refers to a shared memory object, *ftruncate*() shall set the size of the shared memory object to *length*.

If the effect of *ftruncate*() is to decrease the size of a memory mapped file or a shared memory object and whole pages beyond the new end were previously mapped, then the whole pages beyond the new end shall be discarded.

References to discarded pages shall result in the generation of a SIGBUS signal.

If the effect of *ftruncate*() is to increase the size of a memory object, it is unspecified whether the contents of any mapped pages between the old end-of-file and the new are flushed to the underlying object.

RETURN VALUE

Upon successful completion, *ftruncate*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *ftruncate*() function shall fail if:

EINTR
A signal was caught during execution.

EINVAL
The *length* argument was less than 0.

EFBIG or **EINVAL**
The *length* argument was greater than the maximum file size.

EFBIG
The file is a regular file and *length* is greater than the offset maximum established in the open file description associated with *fildes*.

EIO An I/O error occurred while reading from or writing to a file system.

EBADF or **EINVAL**
The *fildes* argument is not a file descriptor open for writing.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), *truncate*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ftrylockfile — stdio locking functions

SYNOPSIS

#include <stdio.h>

int ftrylockfile(FILE **file*);

DESCRIPTION

Refer to *flockfile*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ftw — traverse (walk) a file tree

SYNOPSIS

```
#include <ftw.h>

int ftw(const char *path, int (*fn)(const char *,
   const struct stat *ptr, int flag), int ndirs);
```

DESCRIPTION

The *ftw*() function shall recursively descend the directory hierarchy rooted in *path*. For each object in the hierarchy, *ftw*() shall call the function pointed to by *fn*, passing it a pointer to a null-terminated character string containing the name of the object, a pointer to a **stat** structure containing information about the object, filled in as if *stat*() or *lstat*() had been called to retrieve the information. Possible values of the integer, defined in the *<ftw.h>* header, are:

FTW_D	For a directory.
FTW_DNR	For a directory that cannot be read.
FTW_F	For a non-directory file.
FTW_SL	For a symbolic link (but see also FTW_NS below).
FTW_NS	For an object other than a symbolic link on which *stat*() could not successfully be executed. If the object is a symbolic link and *stat*() failed, it is unspecified whether *ftw*() passes FTW_SL or FTW_NS to the user-supplied function.

If the integer is FTW_DNR, descendants of that directory shall not be processed. If the integer is FTW_NS, the **stat** structure contains undefined values. An example of an object that would cause FTW_NS to be passed to the function pointed to by *fn* would be a file in a directory with read but without execute (search) permission.

The *ftw*() function shall visit a directory before visiting any of its descendants.

The *ftw*() function shall use at most one file descriptor for each level in the tree.

The argument *ndirs* should be in the range [1,{OPEN_MAX}].

The tree traversal shall continue until either the tree is exhausted, an invocation of *fn* returns a non-zero value, or some error, other than **[EACCES]**, is detected within *ftw*().

The *ndirs* argument shall specify the maximum number of directory streams or file descriptors or both available for use by *ftw*() while traversing the tree. When *ftw*() returns it shall close any directory streams and file descriptors it uses not counting any opened by the application-supplied *fn* function.

The results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

The *ftw*() function need not be thread-safe.

RETURN VALUE

If the tree is exhausted, *ftw*() shall return 0. If the function pointed to by *fn* returns a non-zero value, *ftw*() shall stop its tree traversal and return whatever value was returned by the function pointed to by *fn*. If *ftw*() detects an error, it shall return −1 and set *errno* to indicate the error.

If *ftw*() encounters an error other than **[EACCES]** (see FTW_DNR and FTW_NS above), it shall return −1 and set *errno* to indicate the error. The external variable *errno* may contain any error value that is possible when a directory is opened or when one of the *stat* functions is executed on a directory or file.

ERRORS

The *ftw*() function shall fail if:

EACCES
Search permission is denied for any component of *path* or read permission is denied for *path*.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of *path* names an existing file that is neither a directory nor a symbolic link to a directory.

EOVERFLOW
A field in the **stat** structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The *ftw*() function may fail if:

EINVAL
The value of the *ndirs* argument is invalid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

In addition, if the function pointed to by *fn* encounters system errors, *errno* may be set accordingly.

The following sections are informative.

EXAMPLES

Walking a Directory Structure

The following example walks the current directory structure, calling the *fn* function for every directory entry, using at most 10 file descriptors:

```
#include <ftw.h>
...
if (ftw(".", fn, 10) != 0) {
    perror("ftw"); exit(2);
}
```

APPLICATION USAGE

The *ftw*() function may allocate dynamic storage during its operation. If *ftw*() is forcibly terminated, such as by *longjmp*() or *siglongjmp*() being executed by the function pointed to by *fn* or an interrupt routine, *ftw*() does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next invocation.

Applications should use the *nftw*() function instead of the obsolescent *ftw*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *ftw*() function may be removed in a future version.

SEE ALSO

fdopendir( ), *fstatat*( ), *longjmp*( ), *nftw*( ), *siglongjmp*( )

The Base Definitions volume of POSIX.1-2017, **<ftw.h>**, **<sys_stat.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

funlockfile — stdio locking functions

SYNOPSIS

```
#include <stdio.h>

void funlockfile(FILE *file);
```

DESCRIPTION

Refer to *flockfile*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

futimens, utimensat, utimes — set file access and modification times

SYNOPSIS

```
#include <sys/stat.h>

int futimens(int fd, const struct timespec times[2]);

#include <fcntl.h>

int utimensat(int fd, const char *path, const struct timespec times[2],
   int flag);

#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

DESCRIPTION

The *futimens*() and *utimensat*() functions shall set the access and modification times of a file to the values of the *times* argument. The *futimens*() function changes the times of the file associated with the file descriptor *fd*. The *utimensat*() function changes the times of the file pointed to by the *path* argument, relative to the directory associated with the file descriptor *fd*. Both functions allow time specifications accurate to the nanosecond.

For *futimens*() and *utimensat*(), the *times* argument is an array of two **timespec** structures. The first array member represents the date and time of last access, and the second member represents the date and time of last modification. The times in the **timespec** structure are measured in seconds and nanoseconds since the Epoch. The file's relevant timestamp shall be set to the greatest value supported by the file system that is not greater than the specified time.

If the *tv_nsec* field of a **timespec** structure has the special value UTIME_NOW, the file's relevant timestamp shall be set to the greatest value supported by the file system that is not greater than the current time. If the *tv_nsec* field has the special value UTIME_OMIT, the file's relevant timestamp shall not be changed. In either case, the *tv_sec* field shall be ignored.

If the *times* argument is a null pointer, both the access and modification timestamps shall be set to the greatest value supported by the file system that is not greater than the current time. If *utimensat*() is passed a relative path in the *path* argument, the file to be used shall be relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *utimensat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used.

Only a process with the effective user ID equal to the user ID of the file, or with write access to the file, or with appropriate privileges may use *futimens*() or *utimensat*() with a null pointer as the *times* argument or with both *tv_nsec* fields set to the special value UTIME_NOW. Only a process with the effective user ID equal to the user ID of the file or with appropriate privileges may use *futimens*() or *utimensat*() with a non-null *times* argument that does not have both *tv_nsec* fields set to UTIME_NOW and does not have both *tv_nsec* fields set to UTIME_OMIT. If both *tv_nsec* fields are set to UTIME_OMIT, no ownership or permissions check shall be performed for the file, but other error conditions may still be detected (including **[EACCES]** errors related to the path prefix).

Values for the *flag* argument of *utimensat*() are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_SYMLINK_NOFOLLOW
: If *path* names a symbolic link, then the access and modification times of the symbolic link are changed.

Upon successful completion, *futimens*() and *utimensat*() shall mark the last file status change timestamp for update, with the exception that if both *tv_nsec* fields are set to UTIME_OMIT, the file status change timestamp need not be marked for update.

The *utimes*() function shall be equivalent to the *utimensat*() function with the special value AT_FDCWD as the *fd* argument and the *flag* argument set to zero, except that the *times* argument is a **timeval** structure rather than a **timespec** structure, and accuracy is only to the microsecond, not nanosecond, and rounding towards the nearest second may occur.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, the file times shall not be affected.

ERRORS

These functions shall fail if:

EACCES
: The *times* argument is a null pointer, or both *tv_nsec* values are UTIME_NOW, and the effective user ID of the process does not match the owner of the file and write access is denied.

EINVAL
: Either of the *times* argument structures specified a *tv_nsec* value that was neither UTIME_NOW nor UTIME_OMIT, and was a value less than zero or greater than or equal to 1 000 million.

EINVAL
: A new file timestamp would be a value whose *tv_sec* component is not a value supported by the file system.

EPERM
: The *times* argument is not a null pointer, does not have both *tv_nsec* fields set to UTIME_NOW, does not have both *tv_nsec* fields set to UTIME_OMIT, the calling process' effective user ID does not match the owner of the file, and the calling process does not have appropriate privileges.

EROFS
: The file system containing the file is read-only.

The *futimens*() function shall fail if:

EBADF
: The *fd* argument is not a valid file descriptor.

The *utimensat*() function shall fail if:

EACCES
: The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
: The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
: The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

The *utimensat*() and *utimes*() functions shall fail if:

EACCES
: Search permission is denied by a component of the path prefix.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *utimensat*() and *utimes*() functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *utimensat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The purpose of the *utimensat*() function is to set the access and modification time of files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *utimes*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *utimensat*() function it can be guaranteed that the changed file is located relative to the desired directory.

The standard developers considered including a special case for the permissions required by *utimensat*() when one *tv_nsec* field is UTIME_NOW and the other is UTIME_OMIT. One possibility would be to include this case in with the cases where *times* is a null pointer or both fields are UTIME_NOW, where the call is allowed if the process has write permission for the file. However, associating write permission with an update to just the last data access timestamp (which is normally updated by *read*()) did not seem appropriate. The other possibility would be to specify that this one case is allowed if the process has read permission, but this was felt to be too great a departure from the *utime*() and *utimes*() functions on which *utimensat*() is based. If an application needs to set the last data access timestamp to the current time for a file on which it has read permission but is not the owner, it can do so by opening the file, reading one or more bytes (or reading a directory entry, if the file is a directory), and then closing it.

FUTURE DIRECTIONS

None.

SEE ALSO

read( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**, **<sys_time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fwide — set stream orientation

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwide(FILE *stream, int mode);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fwide*() function shall determine the orientation of the stream pointed to by *stream*. If *mode* is greater than zero, the function first attempts to make the stream wide-oriented. If *mode* is less than zero, the function first attempts to make the stream byte-oriented. Otherwise, *mode* is zero and the function does not alter the orientation of the stream.

If the orientation of the stream has already been determined, *fwide*() shall not change it.

The *fwide*() function shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *fwide*(), then check *errno*, and if it is non-zero, assume an error has occurred.

RETURN VALUE

The *fwide*() function shall return a value greater than zero if, after the call, the stream has wide-orientation, a value less than zero if the stream has byte-orientation, or zero if the stream has no orientation.

ERRORS

The *fwide*() function may fail if:

EBADF
The *stream* argument is not a valid stream.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

A call to *fwide*() with *mode* set to zero can be used to determine the current orientation of a stream.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fwprintf, swprintf, wprintf — print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwprintf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swprintf(wchar_t *restrict ws, size_t n,
    const wchar_t *restrict format, ...);
int wprintf(const wchar_t *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fwprintf*() function shall place output on the named output *stream*. The *wprintf*() function shall place output on the standard output stream *stdout*. The *swprintf*() function shall place output followed by the null wide character in consecutive wide characters starting at **ws*; no more than *n* wide characters shall be written, including a terminating null wide character, which is always added (unless *n* is zero).

Each of these functions shall convert, format, and print its arguments under control of the *format* wide-character string. The *format* is composed of zero or more directives: *ordinary wide-characters*, which are simply copied to the output stream, and *conversion specifications*, each of which results in the fetching of zero or more arguments. The results are undefined if there are insufficient arguments for the *format*. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character **%** (see below) is replaced by the sequence **"%n$"**, where *n* is a decimal integer in the range [1,{NL_ARGMAX}], giving the position of the argument in the argument list. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages (see the EXAMPLES section).

The *format* can contain either numbered argument specifications (that is, "%*n*$" and "**m*$"), or unnumbered argument conversion specifications (that is, **%** and *****), but not both. The only exception to this is that **%%** can be mixed with the "%*n*$" form. The results of mixing numbered and unnumbered argument specifications in a *format* wide-character string are undefined. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are specified in the *format* wide-character string.

In *format* wide-character strings containing the "%*n*$" form of conversion specification, numbered arguments in the argument list can be referenced from the *format* wide-character string as many times as required.

In *format* wide-character strings containing the **%** form of conversion specification, each argument in the argument list shall be used exactly once. It is unspecified whether an encoding error occurs if the format string contains **wchar_t** values that do not correspond to members of the character set of the current locale and the specified semantics do not require that value to be processed by *wcrtomb*().

All forms of the *fwprintf*() function allow for the insertion of a locale-dependent radix character in the output string, output as a wide-character value. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (**'.'**).

Each conversion specification is introduced by the **'%'** wide character or by the wide-character sequence "%*n*$", after which the following appear in sequence:

- Zero or more *flags* (in any order), which modify the meaning of the conversion specification.
- An optional minimum *field width*. If the converted value has fewer wide characters than the field width, it shall be padded with <space> characters by default on the left; it shall be padded on the right, if the left-adjustment flag (**'−'**), described below, is given to the field width. The field width takes the form of an <asterisk> (**'*'**), described below, or a decimal integer.
- An optional *precision* that gives the minimum number of digits to appear for the **d**, **i**, **o**, **u**, **x**, and **X** conversion specifiers; the number of digits to appear after the radix character for the **a**, **A**, **e**, **E**, **f**, and **F** conversion specifiers; the maximum number of significant digits for the **g** and **G** conversion specifiers; or the maximum number of wide characters to be printed from a string in the **s** conversion specifiers. The precision takes the form of a <period> (**'.'**) followed either by an <asterisk> (**'*'**), described below, or an optional decimal digit string, where a null digit string is treated as 0. If a precision appears with any other conversion wide character, the behavior is undefined.
- An optional length modifier that specifies the size of the argument.
- A *conversion specifier* wide character that indicates the type of conversion to be applied.

A field width, or precision, or both, may be indicated by an <asterisk> (**'*'**). In this case an argument of type **int** supplies the field width or precision. Applications shall ensure that arguments specifying field width, or precision, or both appear in that order before the argument, if any, to be converted. A negative field width is taken as a **'−'** flag followed by a positive field width. A negative precision is taken as if the precision were omitted. In *format* wide-character strings containing the "%*n*$" form of a conversion specification, a field width or precision may be indicated by the sequence "**m*$", where *m* is a decimal integer in the range [1,{NL_ARGMAX}] giving the position in the argument list (after the *format* argument) of an integer argument containing the field width or precision, for example:

```
wprintf(L"%1$d:%2$.*3$d:%4$.*3$d\n", hour, min, precision, sec);
```

The flag wide characters and their meanings are:

' (The <apostrophe>.) The integer portion of the result of a decimal conversion (**%i**, **%d**, **%u**, **%f**, **%F**, **%g**, or **%G**) shall be formatted with thousands' grouping wide characters. For other conversions, the behavior is undefined. The numeric grouping wide character is used.

− The result of the conversion shall be left-justified within the field. The conversion shall be right-justified if this flag is not specified.

+ The result of a signed conversion shall always begin with a sign (**'+'** or **'−'**). The conversion shall begin with a sign only when a negative value is converted if this flag is not specified.

<space> If the first wide character of a signed conversion is not a sign, or if a signed conversion results in no wide characters, a <space> shall be prefixed to the result. This means that if the <space> and **'+'** flags both appear, the <space> flag shall be ignored.

# Specifies that the value is to be converted to an alternative form. For **o** conversion, it shall increase the precision, if and only if necessary, to force the first digit of the result to be zero (if the value and precision are both 0, a single 0 is printed). For **x** or **X** conversion specifiers, a non-zero result shall have 0x (or 0X) prefixed to it. For **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, the result shall always contain a radix character, even if no digits follow it. Without this flag, a radix character appears in the result of these conversions only if a digit follows it. For **g** and **G** conversion specifiers, trailing zeros shall *not* be removed from the result as they normally are. For other conversion specifiers, the behavior is undefined.

0 For **d**, **i**, **o**, **u**, **x**, **X**, **a**, **A**, **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, leading zeros (following any indication of sign or base) are used to pad to the field width rather than performing space padding, except when converting an infinity or NaN. If the **'0'** and **'−'** flags both appear, the **'0'** flag shall be ignored. For **d**, **i**, **o**, **u**, **x**, and **X** conversion specifiers, if a precision is specified, the **'0'** flag shall be ignored. If the **'0'** and <apostrophe> flags both appear, the grouping wide characters are inserted before zero padding. For other conversions, the behavior is undefined.

The length modifiers and their meanings are:

hh Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **signed char** or **unsigned char** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **signed char** or **unsigned char** before printing); or that a following **n** conversion specifier applies to a pointer to a **signed char** argument.

h Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **short** or **unsigned short** argument (the argument will have been promoted according to the integer promotions, but its value shall be converted to **short** or **unsigned short** before printing); or that a following **n** conversion specifier applies to a pointer to a **short** argument.

l (ell) Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long** or **unsigned long** argument; that a following **n** conversion specifier applies to a pointer to a **long** argument; that a following **c** conversion specifier applies to a **wint_t** argument; that a following **s** conversion specifier applies to a pointer to a **wchar_t** argument; or has no effect on a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier.

ll (ell-ell)
Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **long long** or **unsigned long long** argument; or that a following **n** conversion specifier applies to a pointer to a **long long** argument.

j Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to an **intmax_t** or **uintmax_t** argument; or that a following **n** conversion specifier applies to a pointer to an **intmax_t** argument.

z Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **size_t** or the corresponding signed integer type argument; or that a following **n** conversion specifier applies to a pointer to a signed integer type corresponding to a **size_t** argument.

t Specifies that a following **d**, **i**, **o**, **u**, **x**, or **X** conversion specifier applies to a **ptrdiff_t** or the corresponding **unsigned** type argument; or that a following **n** conversion specifier applies to a pointer to a **ptrdiff_t** argument.

L Specifies that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to a **long double** argument.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The conversion specifiers and their meanings are:

d, i The **int** argument shall be converted to a signed decimal in the style "[−]*dddd*". The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

o The **unsigned** argument shall be converted to unsigned octal format in the style **"dddd"**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

u The **unsigned** argument shall be converted to unsigned decimal format in the style **"dddd"**. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

x The **unsigned** argument shall be converted to unsigned hexadecimal format in the style **"dddd"**; the letters **"abcdef"** are used. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it shall be expanded with leading zeros. The default precision shall be 1. The result of converting zero with an explicit precision of zero shall be no wide characters.

X Equivalent to the **x** conversion specifier, except that letters **"ABCDEF"** are used instead of **"abcdef"**.

f, F The **double** argument shall be converted to decimal notation in the style "[−]*ddd.ddd*", where the number of digits after the radix character shall be equal to the precision specification. If the precision is missing, it shall be taken as 6; if the precision is explicitly zero and no **'#'** flag is present, no radix character shall appear. If a radix character appears, at least one digit shall appear before it. The value shall be rounded in an implementation-defined manner to the appropriate number of digits.

A **double** argument representing an infinity shall be converted in one of the styles **"[-]inf"** or **"[-]infinity"**; which style is implementation-defined. A **double** argument representing a NaN shall be converted in one of the styles **"[-]nan"** or "[−]nan(*n-char-sequence*)"; which style, and the meaning of any *n-char-sequence*, is implementation-defined. The **F** conversion specifier produces **"INF"**, **"INFINITY"**, or **"NAN"** instead of **"inf"**, **"infinity"**, or **"nan"**, respectively.

e, E The **double** argument shall be converted in the style "[−]*d.ddd*e±dd", where there shall be one digit before the radix character (which is non-zero if the argument is non-zero) and the number of digits after it shall be equal to the precision; if the precision is missing, it shall be taken as 6; if the precision is zero and no **'#'** flag is present, no radix character shall appear. The value shall be rounded in an implementation-defined manner to the appropriate number of digits. The **E** conversion wide character shall produce a number with **'E'** instead of **'e'** introducing the exponent. The exponent shall always contain at least two digits. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

g, G The **double** argument representing a floating-point number shall be converted in the style **f** or **e** (or in the style **F** or **E** in the case of a **G** conversion specifier), depending on the value converted and the precision. Let **P** equal the precision if non-zero, 6 if the precision is omitted, or 1 if the precision is zero. Then, if a conversion with style **E** would have an exponent of *X*:

-- If **P**>*X*≥−4, the conversion shall be with style **f** (or **F**) and precision **P**−(*X*+1).

-- Otherwise, the conversion shall be with style **e** (or **E**) and precision **P**−1.

Finally, unless the **'#'** flag is used, any trailing zeros shall be removed from the fractional portion of the result and the decimal-point character shall be removed if there is no fractional portion remaining.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

a, A A **double** argument representing a floating-point number shall be converted in the style "[−]0x*h.hhhh*p±*d*", where there shall be one hexadecimal digit (which is non-zero if the argument is a normalized floating-point number and is otherwise unspecified) before the decimal-point wide character and the number of hexadecimal digits after it shall be equal to the precision; if the precision is missing and FLT_RADIX is a power of 2, then the precision shall be sufficient for an exact representation of the value; if the precision is missing and FLT_RADIX is not a power of 2, then the precision shall be sufficient to distinguish values of type **double**, except that trailing zeros may be omitted; if the precision is zero and the **'#'** flag is not specified, no decimal-point wide character shall appear. The letters **"abcdef"** are used for **a** conversion and the letters **"ABCDEF"** for **A** conversion. The **A** conversion specifier produces a number with **'X'** and **'P'** instead of **'x'** and **'p'**. The exponent shall always contain at least one digit, and only as many more digits as necessary to represent the decimal exponent of 2. If the value is zero, the exponent shall be zero.

A **double** argument representing an infinity or NaN shall be converted in the style of an **f** or **F** conversion specifier.

c If no **l** (ell) qualifier is present, the **int** argument shall be converted to a wide character as if by calling the *btowc*() function and the resulting wide character shall be written. Otherwise, the

wint_t argument shall be converted to **wchar_t**, and written.

s If no **l** (ell) qualifier is present, the application shall ensure that the argument is a pointer to a character array containing a character sequence beginning in the initial shift state. Characters from the array shall be converted as if by repeated calls to the *mbrtowc*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first character is converted, and written up to (but not including) the terminating null wide character. If the precision is specified, no more than that many wide characters shall be written. If the precision is not specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character.

If an **l** (ell) qualifier is present, the application shall ensure that the argument is a pointer to an array of type **wchar_t**. Wide characters from the array shall be written up to (but not including) a terminating null wide character. If no precision is specified, or is greater than the size of the array, the application shall ensure that the array contains a null wide character. If a precision is specified, no more than that many wide characters shall be written.

p The application shall ensure that the argument is a pointer to **void**. The value of the pointer shall be converted to a sequence of printable wide characters in an implementation-defined manner.

n The application shall ensure that the argument is a pointer to an integer into which is written the number of wide characters written to the output so far by this call to one of the *fwprintf*() functions. No argument shall be converted, but one shall be consumed. If the conversion specification includes any flags, a field width, or a precision, the behavior is undefined.

C Equivalent to **lc**.

S Equivalent to **ls**.

% Output a **'%'** wide character; no argument shall be converted. The entire conversion specification shall be **%%**.

If a conversion specification does not match one of the above forms, the behavior is undefined.

In no case does a nonexistent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field shall be expanded to contain the conversion result. Characters generated by *fwprintf*() and *wprintf*() shall be printed as if *fputwc*() had been called.

For **a** and **A** conversions, if FLT_RADIX is not a power of 2 and the result is not exactly representable in the given precision, the result should be one of the two adjacent numbers in hexadecimal floating style with the given precision, with the extra stipulation that the error should have a correct sign for the current rounding direction.

For **e**, **E**, **f**, **F**, **g**, and **G** conversion specifiers, if the number of significant decimal digits is at most DECIMAL_DIG, then the result should be correctly rounded. If the number of significant decimal digits is more than DECIMAL_DIG but the source value is exactly representable with DECIMAL_DIG digits, then the result should be an exact representation with trailing zeros. Otherwise, the source value is bounded by two adjacent decimal strings $L < U$, both having DECIMAL_DIG significant digits; the value of the resultant decimal string D should satisfy $L <= D <= U$, with the extra stipulation that the error should have a correct sign for the current rounding direction.

The last data modification and last file status change timestamps of the file shall be marked for update between the call to a successful execution of *fwprintf*() or *wprintf*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream, or a call to *exit*() or *abort*().

RETURN VALUE

Upon successful completion, these functions shall return the number of wide characters transmitted, excluding the terminating null wide character in the case of *swprintf*(), or a negative value if an output error was encountered, and set *errno* to indicate the error.

If *n* or more wide characters were requested to be written, *swprintf*() shall return a negative value, and set *errno* to indicate the error.

ERRORS

For the conditions under which *fwprintf*() and *wprintf*() fail and may fail, refer to *fputwc*( ).

In addition, all forms of *fwprintf*() shall fail if:

EILSEQ
A wide-character code that does not correspond to a valid character has been detected.

In addition, *fwprintf*() and *wprintf*() may fail if:

ENOMEM
Insufficient storage space is available.

The *swprintf*() shall fail if:

EOVERFLOW
The value of *n* is greater than {INT_MAX} or the number of bytes needed to hold the output excluding the terminating null is greater than {INT_MAX}.

The following sections are informative.

EXAMPLES

To print the language-independent date and time format, the following statement could be used:

```
wprintf(format, weekday, month, day, hour, min);
```

For American usage, *format* could be a pointer to the wide-character string:

```
L"%s, %s %d, %d:%.2d\n"
```

producing the message:

```
Sunday, July 3, 10:02
```

whereas for German usage, *format* could be a pointer to the wide-character string:

```
L"%1$s, %3$d. %2$s, %4$d:%5$.2d\n"
```

producing the message:

```
Sonntag, 3. Juli, 10:02
```

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that there are insufficient arguments for the format, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *btowc*( ), *fputwc*( ), *fwscanf*( ), *mbrtowc*( ), *setlocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fwrite — binary output

SYNOPSIS

```
#include <stdio.h>

size_t fwrite(const void *restrict ptr, size_t size, size_t nitems,
   FILE *restrict stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fwrite*() function shall write, from the array pointed to by *ptr*, up to *nitems* elements whose size is specified by *size*, to the stream pointed to by *stream*. For each object, *size* calls shall be made to the *fputc*() function, taking the values (in order) from an array of **unsigned char** exactly overlaying the object. The file-position indicator for the stream (if defined) shall be advanced by the number of bytes successfully written. If an error occurs, the resulting value of the file-position indicator for the stream is unspecified.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *fwrite*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream, or a call to *exit*() or *abort*().

RETURN VALUE

The *fwrite*() function shall return the number of elements successfully written, which may be less than *nitems* if a write error is encountered. If *size* or *nitems* is 0, *fwrite*() shall return 0 and the state of the stream remains unchanged. Otherwise, if a write error occurs, the error indicator for the stream shall be set, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Because of possible differences in element length and byte ordering, files written using *fwrite*() are application-dependent, and possibly cannot be read using *fread*() by a different application or by the same application on a different processor.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *ferror*( ), *fopen*( ), *fprintf*( ), *putc*( ), *puts*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

fwscanf, swscanf, wscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int fwscanf(FILE *restrict stream, const wchar_t *restrict format, ...);
int swscanf(const wchar_t *restrict ws,
    const wchar_t *restrict format, ...);
int wscanf(const wchar_t *restrict format, ...);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *fwscanf*() function shall read from the named input *stream*. The *wscanf*() function shall read from the standard input stream *stdin*. The *swscanf*() function shall read from the wide-character string *ws*. Each function reads wide characters, interprets them according to a format, and stores the results in its arguments. Each expects, as arguments, a control wide-character string *format* described below, and a set of *pointer* arguments indicating where the converted input should be stored. The result is undefined if there are insufficient arguments for the format. If the *format* is exhausted while arguments remain, the excess arguments are evaluated but are otherwise ignored.

Conversions can be applied to the *n*th argument after the *format* in the argument list, rather than to the next unused argument. In this case, the conversion specifier wide character **%** (see below) is replaced by the sequence **"%n$"**, where *n* is a decimal integer in the range [1,{NL_ARGMAX}]. This feature provides for the definition of *format* wide-character strings that select arguments in an order appropriate to specific languages. In *format* wide-character strings containing the "%*n*$" form of conversion specifications, it is unspecified whether numbered arguments in the argument list can be referenced from the *format* wide-character string more than once.

The *format* can contain either form of a conversion specification—that is, **%** or "%*n*$"— but the two forms cannot normally be mixed within a single *format* wide-character string. The only exception to this is that **%%** or **%*** can be mixed with the "%*n*$" form. When numbered argument specifications are used, specifying the *N*th argument requires that all the leading arguments, from the first to the (*N*−1)th, are pointers.

The *fwscanf*() function in all its forms allows for detection of a language-dependent radix character in the input string, encoded as a wide-character value. The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (**'.'**).

The *format* is a wide-character string composed of zero or more directives. Each directive is composed of one of the following: one or more white-space wide characters (<space>, <tab>, <newline>, <vertical-tab>, or <form-feed>); an ordinary wide character (neither **'%'** nor a white-space character); or a conversion specification. It is unspecified whether an encoding error occurs if the format string contains **wchar_t** values that do not correspond to members of the character set of the current locale and the specified semantics do not require that value to be processed by *wcrtomb*().

Each conversion specification is introduced by the **'%'** or by the character sequence "%*n*$", after which the following appear in sequence:

* An optional assignment-suppressing character **'*'**.

* An optional non-zero decimal integer that specifies the maximum field width.

- An optional assignment-allocation character **'m'**.
- An optional length modifier that specifies the size of the receiving object.
- A conversion specifier wide character that specifies the type of conversion to be applied. The valid conversion specifiers are described below.

The *fwscanf*() functions shall execute each directive of the format in turn. If a directive fails, as detailed below, the function shall return. Failures are described as input failures (due to the unavailability of input bytes) or matching failures (due to inappropriate input).

A directive composed of one or more white-space wide characters is executed by reading input until no more valid input can be read, or up to the first wide character which is not a white-space wide character, which remains unread.

A directive that is an ordinary wide character shall be executed as follows. The next wide character is read from the input and compared with the wide character that comprises the directive; if the comparison shows that they are not equivalent, the directive shall fail, and the differing and subsequent wide characters remain unread. Similarly, if end-of-file, an encoding error, or a read error prevents a wide character from being read, the directive shall fail.

A directive that is a conversion specification defines a set of matching input sequences, as described below for each conversion wide character. A conversion specification is executed in the following steps.

Input white-space wide characters (as specified by *iswspace*( )) shall be skipped, unless the conversion specification includes a **[**, **c**, or **n** conversion specifier.

An item shall be read from the input, unless the conversion specification includes an **n** conversion specifier wide character. An input item is defined as the longest sequence of input wide characters, not exceeding any specified field width, which is an initial subsequence of a matching sequence. The first wide character, if any, after the input item shall remain unread. If the length of the input item is zero, the execution of the conversion specification shall fail; this condition is a matching failure, unless end-of-file, an encoding error, or a read error prevented input from the stream, in which case it is an input failure.

Except in the case of a **%** conversion specifier, the input item (or, in the case of a **%n** conversion specification, the count of input wide characters) shall be converted to a type appropriate to the conversion wide character. If the input item is not a matching sequence, the execution of the conversion specification shall fail; this condition is a matching failure. Unless assignment suppression was indicated by a **'*'**, the result of the conversion shall be placed in the object pointed to by the first argument following the *format* argument that has not already received a conversion result if the conversion specification is introduced by **%**, or in the *n*th argument if introduced by the wide-character sequence "%*n*$". If this object does not have an appropriate type, or if the result of the conversion cannot be represented in the space provided, the behavior is undefined.

The **%c**, **%s**, and **%[** conversion specifiers shall accept an optional assignment-allocation character **'m'**, which shall cause a memory buffer to be allocated to hold the wide-character string converted including a terminating null wide character. In such a case, the argument corresponding to the conversion specifier should be a reference to a pointer value that will receive a pointer to the allocated buffer. The system shall allocate a buffer as if *malloc*() had been called. The application shall be responsible for freeing the memory after usage. If there is insufficient memory to allocate a buffer, the function shall set *errno* to **[ENOMEM]** and a conversion error shall result. If the function returns EOF, any memory successfully allocated for parameters using assignment-allocation character **'m'** by this call shall be freed before the function returns.

The length modifiers and their meanings are:

hh	Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **signed char** or **unsigned char**.
h	Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **short** or **unsigned short**.
l (ell)	Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **long** or **unsigned long**; that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion

specifier applies to an argument with type pointer to **double**; or that a following **c**, **s**, or **[** conversion specifier applies to an argument with type pointer to **wchar_t**. If the **'m'** assignment-allocation character is specified, the conversion applies to an argument with the type pointer to a pointer to **wchar_t**.

ll (ell-ell)
: Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **long long** or **unsigned long long**.

j
: Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **intmax_t** or **uintmax_t**.

z
: Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **size_t** or the corresponding signed integer type.

t
: Specifies that a following **d**, **i**, **o**, **u**, **x**, **X**, or **n** conversion specifier applies to an argument with type pointer to **ptrdiff_t** or the corresponding **unsigned** type.

L
: Specifies that a following **a**, **A**, **e**, **E**, **f**, **F**, **g**, or **G** conversion specifier applies to an argument with type pointer to **long double**.

If a length modifier appears with any conversion specifier other than as specified above, the behavior is undefined.

The following conversion specifier wide characters are valid:

d
: Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *wcstol*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

i
: Matches an optionally signed integer, whose format is the same as expected for the subject sequence of *wcstol*() with 0 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **int**.

o
: Matches an optionally signed octal integer, whose format is the same as expected for the subject sequence of *wcstoul*() with the value 8 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

u
: Matches an optionally signed decimal integer, whose format is the same as expected for the subject sequence of *wcstoul*() with the value 10 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

x
: Matches an optionally signed hexadecimal integer, whose format is the same as expected for the subject sequence of *wcstoul*() with the value 16 for the *base* argument. In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **unsigned**.

a, e, f, g
: Matches an optionally signed floating-point number, infinity, or NaN whose format is the same as expected for the subject sequence of *wcstod*(). In the absence of a size modifier, the application shall ensure that the corresponding argument is a pointer to **float**.

: If the *fwprintf*() family of functions generates character string representations for infinity and NaN (a symbolic entity encoded in floating-point format) to support IEEE Std 754-1985, the *fwscanf*() family of functions shall recognize them as input.

s
: Matches a sequence of non-white-space wide characters. If no **l** (ell) qualifier is present, characters from the input field shall be converted as if by repeated calls to the *wcrtomb*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

If the **l** (ell) qualifier is present and the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character, which shall be added automatically. If the **l** (ell) qualifier is present and the **'m'** assignment-allocation character is present, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

[Matches a non-empty sequence of wide characters from a set of expected wide characters (the *scanset*). If no **l** (ell) qualifier is present, wide characters from the input field shall be converted as if by repeated calls to the *wcrtomb*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to a character array large enough to accept the sequence and the terminating null character, which shall be added automatically. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

If an **l** (ell) qualifier is present and the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to an array of **wchar_t** large enough to accept the sequence and the terminating null wide character. If an **l** (ell) qualifier is present and the **'m'** assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

The conversion specification includes all subsequent wide characters in the *format* string up to and including the matching <right-square-bracket> (**']'**). The wide characters between the square brackets (the *scanlist*) comprise the scanset, unless the wide character after the <left-square-bracket> is a <circumflex> (**'^'**), in which case the scanset contains all wide characters that do not appear in the scanlist between the <circumflex> and the <right-square-bracket>. If the conversion specification begins with **"[]"** or **"[^]"**, the <right-square-bracket> is included in the scanlist and the next <right-square-bracket> is the matching <right-square-bracket> that ends the conversion specification; otherwise, the first <right-square-bracket> is the one that ends the conversion specification. If a **'−'** is in the scanlist and is not the first wide character, nor the second where the first wide character is a **'^'**, nor the last wide character, the behavior is implementation-defined.

c Matches a sequence of wide characters of exactly the number specified by the field width (1 if no field width is present in the conversion specification).

If no **l** (ell) length modifier is present, characters from the input field shall be converted as if by repeated calls to the *wcrtomb*() function, with the conversion state described by an **mbstate_t** object initialized to zero before the first wide character is converted. No null character is added. If the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument is a pointer to the initial element of a character array large enough to accept the sequence. Otherwise, the application shall ensure that the corresponding argument is a pointer to a pointer to a **char**.

No null wide character is added. If an **l** (ell) length modifier is present and the **'m'** assignment-allocation character is not specified, the application shall ensure that the corresponding argument shall be a pointer to the initial element of an array of **wchar_t** large enough to accept the sequence. If an **l** (ell) qualifier is present and the **'m'** assignment-allocation character is specified, the application shall ensure that the corresponding argument is a pointer to a pointer to a **wchar_t**.

p Matches an implementation-defined set of sequences, which shall be the same as the set of sequences that is produced by the **%p** conversion specification of the corresponding *fwprintf*() functions. The application shall ensure that the corresponding argument is a pointer to a pointer to **void**. The interpretation of the input item is implementation-defined. If the input item is a value converted earlier during the same program execution, the pointer that results shall compare equal to that value; otherwise, the behavior of the **%p** conversion is undefined.

n No input is consumed. The application shall ensure that the corresponding argument is a pointer to the integer into which is to be written the number of wide characters read from the input so far by this call to the *fwscanf*() functions. Execution of a **%n** conversion specification shall not increment the assignment count returned at the completion of execution of the function. No argument shall be converted, but one shall be consumed. If the conversion specification includes an assignment-suppressing wide character or a field width, the behavior is undefined.

C Equivalent to **lc**.

S Equivalent to **ls**.

% Matches a single **'%'** wide character; no conversion or assignment shall occur. The complete conversion specification shall be **%%**.

If a conversion specification is invalid, the behavior is undefined.

The conversion specifiers **A**, **E**, **F**, **G**, and **X** are also valid and shall be equivalent to, respectively, **a**, **e**, **f**, **g**, and **x**.

If end-of-file is encountered during input, conversion is terminated. If end-of-file occurs before any wide characters matching the current conversion specification (except for **%n**) have been read (other than leading white-space, where permitted), execution of the current conversion specification shall terminate with an input failure. Otherwise, unless execution of the current conversion specification is terminated with a matching failure, execution of the following conversion specification (if any) shall be terminated with an input failure.

Reaching the end of the string in *swscanf*() shall be equivalent to encountering end-of-file for *fwscanf*().

If conversion terminates on a conflicting input, the offending input shall be left unread in the input. Any trailing white space (including <newline>) shall be left unread unless matched by a conversion specification. The success of literal matches and suppressed assignments is only directly determinable via the **%n** conversion specification.

The *fwscanf*() and *wscanf*() functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetwc*(), *fgetws*(), *fwscanf*(), *getwc*(), *getwchar*(), *vfwscanf*(), *vwscanf*(), or *wscanf*() using *stream* that returns data not supplied by a prior call to *ungetwc*().

RETURN VALUE

Upon successful completion, these functions shall return the number of successfully matched and assigned input items; this number can be zero in the event of an early matching failure. If the input ends before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned. If an error occurs before the first conversion (if any) has completed, and without a matching failure having occurred, EOF shall be returned and *errno* shall be set to indicate the error. If a read error occurs, the error indicator for the stream shall be set.

ERRORS

For the conditions under which the *fwscanf*() functions shall fail and may fail, refer to *fgetwc*( ).

In addition, the *fwscanf*() function shall fail if:

EILSEQ
Input byte sequence does not form a valid character.

ENOMEM
Insufficient storage space is available.

In addition, the *fwscanf*() function may fail if:

EINVAL
There are insufficient arguments.

The following sections are informative.

EXAMPLES

The call:

```
int i, n; float x; char name[50];
n = wscanf(L"%d%f%s", &i, &x, name);
```

with the input line:

```
25 54.32E-1 Hamster
```

assigns to *n* the value 3, to *i* the value 25, to *x* the value 5.432, and *name* contains the string **"Hamster"**.

The call:

```
int i; float x; char name[50];
(void) wscanf(L"%2d%f%*d %[0123456789]", &i, &x, name);
```

with input:

```
56789 0123 56a72
```

assigns 56 to *i*, 789.0 to *x*, skips 0123, and places the string **"56\0"** in *name*. The next call to *getchar*() shall return the character **'a'**.

APPLICATION USAGE

In format strings containing the **'%'** form of conversion specifications, each argument in the argument list is used exactly once.

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *fwscanf*(), this is memory allocated via use of the **'m'** assignment-allocation character.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *getwc*( ), *fwprintf*( ), *setlocale*( ), *wcstod*( ), *wcstol*( ), *wcstoul*( ), *wcrtomb*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<inttypes.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gai_strerror — address and name information error description

SYNOPSIS

#include <netdb.h>

const char *gai_strerror(int *ecode*);

DESCRIPTION

The *gai_strerror*() function shall return a text string describing an error value for the *getaddrinfo*() and *getnameinfo*() functions listed in the *<netdb.h>* header.

When the *ecode* argument is one of the following values listed in the *<netdb.h>* header:

tab(!); le le. T{
[EAI_AGAIN]
[EAI_BADFLAGS]
[EAI_FAIL]
[EAI_FAMILY]
[EAI_MEMORY]
T}!T{
[EAI_NONAME]
[EAI_OVERFLOW]
[EAI_SERVICE]
[EAI_SOCKTYPE]
[EAI_SYSTEM]
T}

the function return value shall point to a string describing the error. If the argument is not one of those values, the function shall return a pointer to a string whose contents indicate an unknown error.

RETURN VALUE

Upon successful completion, *gai_strerror*() shall return a pointer to an implementation-defined string.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

freeaddrinfo( )

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getaddrinfo — get address information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

int getaddrinfo(const char *restrict nodename,
   const char *restrict servname,
   const struct addrinfo *restrict hints,
   struct addrinfo **restrict res);
```

DESCRIPTION

Refer to *freeaddrinfo*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getc — get a byte from a stream

SYNOPSIS

#include <stdio.h>

int getc(FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *getc*() function shall be equivalent to *fgetc*( ), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fgetc*( ).

ERRORS

Refer to *fgetc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *getc*() is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

Since it may be implemented as a macro, *getc*() may treat incorrectly a *stream* argument with side-effects. In particular, *getc*(**f*++) does not necessarily work as expected. Therefore, use of this function should be preceded by **"#undef**getc" in such situations; *fgetc*() could also be used.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getc_unlocked, getchar_unlocked, putc_unlocked, putchar_unlocked — stdio with explicit client locking

SYNOPSIS

```
#include <stdio.h>

int getc_unlocked(FILE *stream);
int getchar_unlocked(void);
int putc_unlocked(int c, FILE *stream);
int putchar_unlocked(int c);
```

DESCRIPTION

Versions of the functions *getc*(), *getchar*(), *putc*(), and *putchar*() respectively named *getc_unlocked*(), *getchar_unlocked*(), *putc_unlocked*(), and *putchar_unlocked*() shall be provided which are functionally equivalent to the original versions, with the exception that they are not required to be implemented in a fully thread-safe manner. They shall be thread-safe when used within a scope protected by *flockfile*() (or *ftrylockfile*()) and *funlockfile*(). These functions can safely be used in a multi-threaded program if and only if they are called while the invoking thread owns the (**FILE ***) object, as is the case after a successful call to the *flockfile*() or *ftrylockfile*() functions.

If *getc_unlocked*() or *putc_unlocked*() are implemented as macros they may evaluate *stream* more than once, so the *stream* argument should never be an expression with side-effects.

RETURN VALUE

See *getc*( ), *getchar*( ), *putc*( ), and *putchar*( ).

ERRORS

See *getc*( ), *getchar*( ), *putc*( ), and *putchar*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since they may be implemented as macros, *getc_unlocked*() and *putc_unlocked*() may treat incorrectly a *stream* argument with side-effects. In particular, *getc_unlocked*(*f++) and *putc_unlocked*(c,*f++) do not necessarily work as expected. Therefore, use of these functions in such situations should be preceded by the following statement as appropriate:

```
#undef getc_unlocked
#undef putc_unlocked
```

RATIONALE

Some I/O functions are typically implemented as macros for performance reasons (for example, *putc*() and *getc*()). For safety, they need to be synchronized, but it is often too expensive to synchronize on every character. Nevertheless, it was felt that the safety concerns were more important; consequently, the *getc*(), *getchar*(), *putc*(), and *putchar*() functions are required to be thread-safe. However, unlocked versions are also provided with names that clearly indicate the unsafe nature of their operation but can be used to exploit their higher performance. These unlocked versions can be safely used only within explicitly locked program regions, using exported locking primitives. In particular, a sequence such as:

```
flockfile(fileptr);
putc_unlocked('1', fileptr);
putc_unlocked('\n', fileptr);
fprintf(fileptr, "Line 2\n");
funlockfile(fileptr);
```

is permissible, and results in the text sequence:

```

1
Line 2
```

being printed without being interspersed with output from other threads.

It would be wrong to have the standard names such as *getc*(), *putc*(), and so on, map to the ‘‘faster, but unsafe’’ rather than the ‘‘slower, but safe’’ versions. In either case, you would still want to inspect all uses of *getc*(), *putc*(), and so on, by hand when converting existing code. Choosing the safe bindings as the default, at least, results in correct code and maintains the ‘‘atomicity at the function’’ invariant. To do otherwise would introduce gratuitous synchronization errors into converted code. Other routines that modify the *stdio* (**FILE ***) structures or buffers are also safely synchronized.

Note that there is no need for functions of the form *getc_locked*(), *putc_locked*(), and so on, since this is the functionality of *getc*(), *putc*(), *et al*. It would be inappropriate to use a feature test macro to switch a macro definition of *getc*() between *getc_locked*() and *getc_unlocked*(), since the ISO C standard requires an actual function to exist, a function whose behavior could not be changed by the feature test macro. Also, providing both the *xxx_locked*() and *xxx_unlocked*() forms leads to the confusion of whether the suffix describes the behavior of the function or the circumstances under which it should be used.

Three additional routines, *flockfile*(), *ftrylockfile*(), and *funlockfile*() (which may be macros), are provided to allow the user to delineate a sequence of I/O statements that are executed synchronously.

The *ungetc*() function is infrequently called relative to the other functions/macros so no unlocked variation is needed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *flockfile*( ), *getc*( ), *getchar*( ), *putc*( ), *putchar*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getchar — get a byte from a *stdin* stream

SYNOPSIS

```
#include <stdio.h>

int getchar(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *getchar*() function shall be equivalent to *getc*(*stdin*).

RETURN VALUE

Refer to *fgetc*( ).

ERRORS

Refer to *fgetc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the integer value returned by *getchar*() is stored into a variable of type **char** and then compared against the integer constant EOF, the comparison may never succeed, because sign-extension of a variable of type **char** on widening to integer is implementation-defined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *getc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getchar_unlocked — stdio with explicit client locking

SYNOPSIS

```
#include <stdio.h>

int getchar_unlocked(void);
```

DESCRIPTION

Refer to *getc_unlocked*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getcwd — get the pathname of the current working directory

SYNOPSIS

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
```

DESCRIPTION

The *getcwd*() function shall place an absolute pathname of the current working directory in the array pointed to by *buf*, and return *buf*. The pathname shall contain no components that are dot or dot-dot, or are symbolic links.

If there are multiple pathnames that *getcwd*() could place in the array pointed to by *buf*, one beginning with a single <slash> character and one or more beginning with two <slash> characters, then *getcwd*() shall place the pathname beginning with a single <slash> character in the array. The pathname shall not contain any unnecessary <slash> characters after the leading one or two <slash> characters.

The *size* argument is the size in bytes of the character array pointed to by the *buf* argument. If *buf* is a null pointer, the behavior of *getcwd*() is unspecified.

RETURN VALUE

Upon successful completion, *getcwd*() shall return the *buf* argument. Otherwise, *getcwd*() shall return a null pointer and set *errno* to indicate the error. The contents of the array pointed to by *buf* are then undefined.

ERRORS

The *getcwd*() function shall fail if:

EINVAL
The *size* argument is 0.

ERANGE
The *size* argument is greater than 0, but is smaller than the length of the string +1.

The *getcwd*() function may fail if:

EACCES
Search permission was denied for the current directory, or read or search permission was denied for a directory above the current directory in the file hierarchy.

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

The following example uses {PATH_MAX} as the initial buffer size (unless it is indeterminate or very large), and calls *getcwd*() with progressively larger buffers until it does not give an **[ERANGE]** error.

```
#include <stdlib.h>
#include <errno.h>
#include <unistd.h>

...

long path_max;
size_t size;
char *buf;
char *ptr;
```

```
path_max = pathconf(".", _PC_PATH_MAX);
if (path_max == -1)
    size = 1024;
else if (path_max > 10240)
    size = 10240;
else
    size = path_max;

for (buf = ptr = NULL; ptr == NULL; size *= 2)
{
    if ((buf = realloc(buf, size)) == NULL)
    {
        ... handle error ...
    }

    ptr = getcwd(buf, size);
    if (ptr == NULL && errno != ERANGE)
    {
        ... handle error ...
    }
}
...
free (buf);
```

APPLICATION USAGE

If the pathname obtained from *getcwd*() is longer than {PATH_MAX} bytes, it could produce an **[ENAMETOOLONG]** error if passed to *chdir*(). Therefore, in order to return to that directory it may be necessary to break the pathname into sections shorter than {PATH_MAX} bytes and call *chdir*() on each section in turn (the first section being an absolute pathname and subsequent sections being relative pathnames). A simpler way to handle saving and restoring the working directory when it may be deeper than {PATH_MAX} bytes in the file hierarchy is to use a file descriptor and *fchdir*(), rather than *getcwd*() and *chdir*(). However, the two methods do have some differences. The *fchdir*() approach causes the program to restore a working directory even if it has been renamed in the meantime, whereas the *chdir*() approach restores to a directory with the same name as the original, even if the directories were renamed in the meantime. Since the *fchdir*() approach does not access parent directories, it can succeed when *getcwd*() would fail due to permissions problems. In applications conforming to earlier versions of this standard, it was not possible to use the *fchdir*() approach when the working directory is searchable but not readable, as the only way to open a directory was with O_RDONLY, whereas the *getcwd*() approach can succeed in this case.

RATIONALE

Having *getcwd*() take no arguments and instead use the *malloc*() function to produce space for the returned argument was considered. The advantage is that *getcwd*() knows how big the working directory pathname is and can allocate an appropriate amount of space. But the programmer would have to use the *free*() function to free the resulting object, or each use of *getcwd*() would further reduce the available memory. Finally, *getcwd*() is taken from the SVID where it has the two arguments used in this volume of POSIX.1-2017.

The older function *getwd*( ) was rejected for use in this context because it had only a buffer argument and no *size* argument, and thus had no way to prevent overwriting the buffer, except to depend on the programmer to provide a large enough buffer.

On some implementations, if *buf* is a null pointer, *getcwd*() may obtain *size* bytes of memory using *malloc*(). In this case, the pointer returned by *getcwd*() may be used as the argument in a subsequent call to *free*(). Invoking *getcwd*() with *buf* as a null pointer is not recommended in conforming applications.

Earlier implementations of *getcwd*() sometimes generated pathnames like **"../../../subdirname"** internally, using them to explore the path of ancestor directories back to the root. If one of these internal pathnames exceeded {PATH_MAX} in length, the implementation could fail with *errno* set to **[ENAMETOOLONG]**. This is no longer allowed.

If a program is operating in a directory where some (grand)parent directory does not permit reading, *getcwd*() may fail, as in most implementations it must read the directory to determine the name of the file. This can occur if search, but not read, permission is granted in an intermediate directory, or if the program is placed in that directory by some more privileged process (for example, login). Including the **[EACCES]** error condition makes the reporting of the error consistent and warns the application developer that *getcwd*() can fail for reasons beyond the control of the application developer or user. Some implementations can avoid this occurrence (for example, by implementing *getcwd*() using *pwd*, where *pwd* is a set-user-root process), thus the error was made optional. Since this volume of POSIX.1-2017 permits the addition of other errors, this would be a common addition and yet one that applications could not be expected to deal with without this addition.

FUTURE DIRECTIONS

None.

SEE ALSO

malloc( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getdate — convert user format date and time

SYNOPSIS

#include <time.h>

struct tm *getdate(const char **string*);

DESCRIPTION

The *getdate*() function shall convert a string representation of a date or time into a broken-down time.

The external variable or macro *getdate_err*, which has type **int**, is used by *getdate*() to return error values. It is unspecified whether *getdate_err* is a macro or an identifier declared with external linkage, and whether or not it is a modifiable lvalue. If a macro definition is suppressed in order to access an actual object, or a program defines an identifier with the name *getdate_err*, the behavior is undefined.

Templates are used to parse and interpret the input string. The templates are contained in a text file identified by the environment variable *DATEMSK*. The *DATEMSK* variable should be set to indicate the full pathname of the file that contains the templates. The first line in the template that matches the input specification is used for interpretation and conversion into the internal time format.

The following conversion specifications shall be supported:

%%	Equivalent to **%**.
%a	Abbreviated weekday name.
%A	Full weekday name.
%b	Abbreviated month name.
%B	Full month name.
%c	Locale's appropriate date and time representation.
%C	Century number [00,99]; leading zeros are permitted but not required.
%d	Day of month [01,31]; the leading 0 is optional.
%D	Date as **%m/%d/%y**.
%e	Equivalent to **%d**.
%h	Abbreviated month name.
%H	Hour [00,23].
%I	Hour [01,12].
%m	Month number [01,12].
%M	Minute [00,59].
%n	Equivalent to <newline>.
%p	Locale's equivalent of either AM or PM.
%r	The locale's appropriate representation of time in AM and PM notation. In the POSIX locale, this shall be equivalent to **%I:%M:%S %p**.
%R	Time as **%H:%M**.
%S	Seconds [00,60]. The range goes to 60 (rather than stopping at 59) to allow positive leap seconds to be expressed. Since leap seconds cannot be predicted by any algorithm, leap second data must come from some external source.

%t	Equivalent to <tab>.
%T	Time as **%H**:**%M**:**%S**.
%w	Weekday number (Sunday = [0,6]).
%x	Locale's appropriate date representation.
%X	Locale's appropriate time representation.
%y	Year within century. When a century is not otherwise specified, values in the range [69,99] shall refer to years 1969 to 1999 inclusive, and values in the range [00,68] shall refer to years 2000 to 2068 inclusive. **Note:** It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
%Y	Year as **"ccyy"** (for example, 2001).
%Z	Timezone name or no characters if no timezone exists. If the timezone supplied by **%Z** is not the timezone that *getdate*() expects, an invalid input specification error shall result. The *getdate*() function calculates an expected timezone based on information supplied to the function (such as the hour, day, and month).

The match between the template and input specification performed by *getdate*() shall be case-insensitive.

The month and weekday names can consist of any combination of upper and lowercase letters. The process can request that the input date or time specification be in a specific language by setting the *LC_TIME* category (see *setlocale*( )).

Leading zeros are not necessary for the descriptors that allow leading zeros. However, at most two digits are allowed for those descriptors, including leading zeros. Extra white space in either the template file or in *string* shall be ignored.

The results are undefined if the conversion specifications **%c**, **%x**, and **%X** include unsupported conversion specifications.

The following rules apply for converting the input specification into the internal format:

* If **%Z** is being scanned, then *getdate*() shall initialize the broken-down time to be the current time in the scanned timezone. Otherwise, it shall initialize the broken-down time based on the current local time as if *localtime*() had been called.
* If only the weekday is given, the day chosen shall be the day, starting with today and moving into the future, which first matches the named day.
* If only the month (and no year) is given, the month chosen shall be the month, starting with the current month and moving into the future, which first matches the named month. The first day of the month shall be assumed if no day is given.
* If no hour, minute, and second are given, the current hour, minute, and second shall be assumed.
* If no date is given, the hour chosen shall be the hour, starting with the current hour and moving into the future, which first matches the named hour.

If a conversion specification in the DATEMSK file does not correspond to one of the conversion specifications above, the behavior is unspecified.

The *getdate*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *getdate*() shall return a pointer to a **struct tm**. Otherwise, it shall return a null pointer and set *getdate_err* to indicate the error.

ERRORS

The *getdate*() function shall fail in the following cases, setting *getdate_err* to the value shown in the list below. Any changes to *errno* are unspecified.

1. The *DATEMSK* environment variable is null or undefined.
2. The template file cannot be opened for reading.
3. Failed to get file status information.
4. The template file is not a regular file.
5. An I/O error is encountered while reading the template file.
6. Memory allocation failed (not enough memory available).
7. There is no line in the template that matches the input.
8. Invalid input specification. For example, February 31; or a time is specified that cannot be represented in a **time_t** (representing the time in seconds since the Epoch).

The following sections are informative.

EXAMPLES

1. The following example shows the possible contents of a template:

```
%m
%A %B %d, %Y, %H:%M:%S
%A
%B
%m/%d/%y %I %p
%d,%m,%Y %H:%M
at %A the %dst of %B in %Y
run job at %I %p,%B %dnd
%A den %d. %B %Y %H.%M Uhr
```

2. The following are examples of valid input specifications for the template in Example 1:

```
getdate("10/1/87 4 PM");
getdate("Friday");
getdate("Friday September 18, 1987, 10:30:30");
getdate("24,9,1986 10:30");
getdate("at monday the 1st of december in 1986");
getdate("run job at 3 PM, december 2nd");
```

If the *LC_TIME* category is set to a German locale that includes *freitag* as a weekday name and *oktober* as a month name, the following would be valid:

```
getdate("freitag den 10. oktober 1986 10.30 Uhr");
```

3. The following example shows how local date and time specification can be defined in the template:

box tab(!) center; cB | cB lf5 | lf5. Invocation!Line in Template _ getdate("11/27/86")!%m/%d/%y getdate("27.11.86")!%d.%m.%y getdate("86-11-27")!%y-%m-%d getdate("Friday 12:00:00")!%A %H:%M:%S

4. The following examples help to illustrate the above rules assuming that the current date is Mon Sep 22 12:19:47 EDT 1986 and the *LC_TIME* category is set to the default C or POSIX locale:

box tab(!) center; cB | cB | cB lf5 | lf5 | l. Input!Line in Template!Date _ Mon!%a!Mon Sep 22 12:19:47 EDT 1986 Sun!%a!Sun Sep 28 12:19:47 EDT 1986 Fri!%a!Fri Sep 26 12:19:47 EDT 1986 September!%B!Mon Sep 1 12:19:47 EDT 1986 January!%B!Thu Jan 1 12:19:47 EST 1987 December!%B!Mon Dec 1 12:19:47 EST 1986 Sep Mon!%b %a!Mon Sep 1 12:19:47 EDT 1986 Jan Fri!%b %a!Fri Jan 2 12:19:47 EST 1987 Dec Mon!%b %a!Mon Dec 1 12:19:47 EST 1986 Jan Wed 1989!%b

%a %Y!Wed Jan 4 12:19:47 EST 1989 Fri 9!%a %H!Fri Sep 26 09:00:00 EDT 1986 Feb 10:30!%b %H:%S!Sun Feb 1 10:00:30 EST 1987 10:30!%H:%M!Tue Sep 23 10:30:00 EDT 1986 13:30!%H:%M!Mon Sep 22 13:30:00 EDT 1986

APPLICATION USAGE

Although historical versions of *getdate*() did not require that <*time.h*> declare the external variable *getdate_err*, this volume of POSIX.1-2017 does require it. The standard developers encourage applications to remove declarations of *getdate_err* and instead incorporate the declaration by including <*time.h*>.

Applications should use **%Y** (4-digit years) in preference to **%y** (2-digit years).

RATIONALE

In standard locales, the conversion specifications **%c**, **%x**, and **%X** do not include unsupported conversion specifiers and so the text regarding results being undefined is not a problem in that case.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime( ), *localtime*( ), *setlocale*( ), *strftime*( ), *times*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getdelim, getline — read a delimited record from *stream*

SYNOPSIS

```
#include <stdio.h>

ssize_t getdelim(char **restrict lineptr, size_t *restrict n,
   int delimiter, FILE *restrict stream);
ssize_t getline(char **restrict lineptr, size_t *restrict n,
   FILE *restrict stream);
```

DESCRIPTION

The *getdelim*() function shall read from *stream* until it encounters a character matching the *delimiter* character. The *delimiter* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** of equal value that terminates the read process. If the *delimiter* argument has any other value, the behavior is undefined.

The application shall ensure that **lineptr* is a valid argument that could be passed to the *free*() function. If **n* is non-zero, the application shall ensure that **lineptr* either points to an object of size at least **n* bytes, or is a null pointer.

If **lineptr* is a null pointer or if the object pointed to by **lineptr* is of insufficient size, an object shall be allocated as if by *malloc*() or the object shall be reallocated as if by *realloc*(), respectively, such that the object is large enough to hold the characters to be written to it, including the terminating NUL, and **n* shall be set to the new size. If the object was allocated, or if the reallocation operation moved the object, **lineptr* shall be updated to point to the new object or new location. The characters read, including any delimiter, shall be stored in the object, and a terminating NUL added when the delimiter or end-of-file is encountered.

The *getline*() function shall be equivalent to the *getdelim*() function with the *delimiter* character equal to the <newline> character.

The *getdelim*() and *getline*() functions may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *fscanf*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, the *getline*() and *getdelim*() functions shall return the number of bytes written into the buffer, including the delimiter character if one was encountered before EOF, but excluding the terminating NUL character. If the end-of-file indicator for the stream is set, or if no characters were read and the stream is at end-of-file, the end-of-file indicator for the stream shall be set and the function shall return −1. If an error occurs, the error indicator for the stream shall be set, and the function shall return −1 and set *errno* to indicate the error.

ERRORS

For the conditions under which the *getdelim*() and *getline*() functions shall fail and may fail, refer to *fgetc*( ).

In addition, these functions shall fail if:

EINVAL
lineptr or *n* is a null pointer.

ENOMEM
Insufficient memory is available.

These functions may fail if:

EOVERFLOW
The number of bytes to be written into the buffer, including the delimiter character (if encountered), would exceed {SSIZE_MAX}.

The following sections are informative.

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp;
    char *line = NULL;
    size_t len = 0;
    ssize_t read;
    fp = fopen("/etc/motd", "r");
    if (fp == NULL)
        exit(1);
    while ((read = getline(&line, &len, fp)) != -1) {
        printf("Retrieved line of length %zu :\n", read);
        printf("%s", line);
    }
    if (ferror(fp)) {
        /* handle error */
    }
    free(line);
    fclose(fp);
    return 0;
}
```

APPLICATION USAGE

Setting **lineptr* to a null pointer and **n* to zero are allowed and a recommended way to start parsing a file.

The *ferror*() or *feof*() functions should be used to distinguish between an error condition and an end-of-file condition.

Although a NUL terminator is always supplied after the line, note that *strlen*(**lineptr*) will be smaller than the return value if the line contains embedded NUL characters.

RATIONALE

These functions are widely used to solve the problem that the *fgets*() function has with long lines. The functions automatically enlarge the target buffers if needed. These are especially useful since they reduce code needed for applications.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetc*( ), *fgets*( ), *free*( ), *malloc*( ), *realloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getegid — get the effective group ID

SYNOPSIS

```
#include <unistd.h>

gid_t getegid(void);
```

DESCRIPTION

The *getegid*() function shall return the effective group ID of the calling process. The *getegid*() function shall not modify *errno*.

RETURN VALUE

The *getegid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getegid*() will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**gid_t**)−1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

geteuid( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getenv — get value of an environment variable

SYNOPSIS

```
#include <stdlib.h>

char *getenv(const char *name);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *getenv*() function shall search the environment of the calling process (see the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*) for the environment variable *name* if it exists and return a pointer to the value of the environment variable. If the specified environment variable cannot be found, a null pointer shall be returned. The application shall ensure that it does not modify the string pointed to by the *getenv*() function.

The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to *getenv*(), *setenv*(), *unsetenv*(),
or (if supported) *putenv*() but they shall not be affected by a call to any other function in this volume of POSIX.1-2017.

The returned string pointer might also be invalidated if the calling thread is terminated.

The *getenv*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *getenv*() shall return a pointer to a string containing the *value* for the specified *name*. If the specified *name* cannot be found in the environment of the calling process, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Getting the Value of an Environment Variable

The following example gets the value of the *HOME* environment variable.

```
#include <stdlib.h>
...
const char *name = "HOME";
char *value;

value = getenv(name);
```

APPLICATION USAGE

None.

RATIONALE

The *clearenv*() function was considered but rejected. The *putenv*() function has now been included for alignment with the Single UNIX Specification.

The *getenv*() function is inherently not thread-safe because it returns a value pointing to static data.

Conforming applications are required not to directly modify the pointers to which *environ* points, but to use only the *setenv*(), *unsetenv*(), and *putenv*() functions, or assignment to *environ* itself, to manipulate the process environment. This constraint allows the implementation to properly manage the memory it allocates. This enables the implementation to free any space it has allocated to strings (and perhaps the pointers to them) stored in *environ* when *unsetenv*() is called. A C runtime start-up procedure (that which invokes *main*() and perhaps initializes *environ*) can also initialize a flag indicating that none of the environment has yet been copied to allocated storage, or that the separate table has not yet been initialized. If the application switches to a complete new environment by assigning a new value to *environ*, this can be detected by *getenv*(), *setenv*(), *unsetenv*(), or *putenv*() and the implementation can at that point reinitialize based on the new environment. (This may include copying the environment strings into a new array and assigning *environ* to point to it.)

In fact, for higher performance of *getenv*(), implementations that do not provide *putenv*() could also maintain a separate copy of the environment in a data structure that could be searched much more quickly (such as an indexed hash table, or a binary tree), and update both it and the linear list at *environ* when *setenv*() or *unsetenv*() is invoked. On implementations that do provide *putenv*(), such a copy might still be worthwhile but would need to allow for the fact that applications can directly modify the content of environment strings added with *putenv*(). For example, if an environment string found by searching the copy is one that was added using *putenv*(), the implementation would need to check that the string in *environ* still has the same name (and value, if the copy includes values), and whenever searching the copy produces no match the implementation would then need to search each environment string in *environ* that was added using *putenv*() in case any of them have changed their names and now match. Thus, each use of *putenv*() to add to the environment would reduce the speed advantage of having the copy.

Performance of *getenv*() can be important for applications which have large numbers of environment variables. Typically, applications like this use the environment as a resource database of user-configurable parameters. The fact that these variables are in the user's shell environment usually means that any other program that uses environment variables (such as *ls*, which attempts to use *COLUMNS*), or really almost any utility (*LANG*, *LC_ALL*, and so on) is similarly slowed down by the linear search through the variables.

An implementation that maintains separate data structures, or even one that manages the memory it consumes, is not currently required as it was thought it would reduce consensus among implementors who do not want to change their historical implementations.

FUTURE DIRECTIONS

A future version may add one or more functions to access and modify the environment in a thread-safe manner.

SEE ALSO

exec, *putenv*( ), *setenv*( ), *unsetenv*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

geteuid — get the effective user ID

SYNOPSIS

```
#include <unistd.h>

uid_t geteuid(void);
```

DESCRIPTION

The *geteuid*() function shall return the effective user ID of the calling process. The *geteuid*() function shall not modify *errno*.

RETURN VALUE

The *geteuid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *geteuid*() will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**uid_t**)−1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getgid — get the real group ID

SYNOPSIS

```
#include <unistd.h>

gid_t getgid(void);
```

DESCRIPTION

The *getgid*() function shall return the real group ID of the calling process. The *getgid*() function shall not modify *errno*.

RETURN VALUE

The *getgid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getgid*() will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**gid_t**)−1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getgrent — get the group database entry

SYNOPSIS

```
#include <grp.h>

struct group *getgrent(void);
```

DESCRIPTION

Refer to *endgrent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getgrgid, getgrgid_r — get group database entry for a group ID

SYNOPSIS

```
#include <grp.h>

struct group *getgrgid(gid_t gid);
int getgrgid_r(gid_t gid, struct group *grp, char *buffer,
   size_t bufsize, struct group **result);
```

DESCRIPTION

The *getgrgid*() function shall search the group database for an entry with a matching *gid*.

The *getgrgid*() function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getgrgid*(). If *get-grgid*() returns a null pointer and *errno* is set to non-zero, an error occurred.

The *getgrgid_r*() function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *gid*. Storage referenced by the group structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf*(_SC_GETGR_R_SIZE_MAX) returns either −1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

Upon successful completion, *getgrgid*() shall return a pointer to a **struct group** with the structure defined in *<grp.h>* with a matching entry if one is found. The *getgrgid*() function shall return a null pointer if either the requested entry was not found, or an error occurred. If the requested entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getgrent*(), *get-grgid*(), or *getgrnam*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

If successful, the *getgrgid_r*() function shall return zero; otherwise, an error number shall be returned to in-dicate the error.

ERRORS

The *getgrgid*() and *getgrgid_r*() functions may fail if:

EIO An I/O error has occurred.

EINTR
A signal was caught during *getgrgid*().

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The *getgrgid_r*() function may fail if:

ERANGE
Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

The following sections are informative.

EXAMPLES

Note that *sysconf*(_SC_GETGR_R_SIZE_MAX) may return −1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getgrid_r*().

```
long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrgid_r(42, &result, buffer, len, &resultp)) == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Finding an Entry in the Group Database

The following example uses *getgrgid*() to search the group database for a group ID that was previously stored in a **stat** structure, then prints out the group name if it is found. If the group is not found, the program prints the numeric value of the group for the entry.

```
#include <sys/types.h>
#include <grp.h>
#include <stdio.h>
...
struct stat statbuf;
struct group *grp;
...
if ((grp = getgrgid(statbuf.st_gid)) != NULL)
    printf(" %-8.8s", grp->gr_name);
else
    printf(" %-8d", statbuf.st_gid);
...
```

APPLICATION USAGE

The *getgrgid_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return −1 from *sysconf*() indicating that there is no maximum for _SC_GETGR_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrnam*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<grp.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getgrnam, getgrnam_r — search group database for a name

SYNOPSIS

```
#include <grp.h>

struct group *getgrnam(const char *name);
int getgrnam_r(const char *name, struct group *grp, char *buffer,
   size_t bufsize, struct group **result);
```

DESCRIPTION

The *getgrnam*() function shall search the group database for an entry with a matching *name*.

The *getgrnam*() function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getgrnam*(). If *getgrnam*() returns a null pointer and *errno* is set to non-zero, an error occurred.

The *getgrnam_r*() function shall update the **group** structure pointed to by *grp* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the group database with a matching *name*. Storage referenced by the **group** structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf*(_SC_GETGR_R_SIZE_MAX) returns either −1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer is returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The *getgrnam*() function shall return a pointer to a **struct group** with the structure defined in *<grp.h>* with a matching entry if one is found. The *getgrnam*() function shall return a null pointer if either the requested entry was not found, or an error occurred. If the requested entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getgrent*(), *getgrgid*(), or *getgrnam*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

The *getgrnam_r*() function shall return zero on success or if the requested entry was not found and no error has occurred. If any error has occurred, an error number shall be returned to indicate the error.

ERRORS

The *getgrnam*() and *getgrnam_r*() functions may fail if:

EIO An I/O error has occurred.

EINTR
A signal was caught during *getgrnam*().

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The *getgrnam_r*() function may fail if:

ERANGE
Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **group** structure.

The following sections are informative.

EXAMPLES

Note that *sysconf*(_SC_GETGR_R_SIZE_MAX) may return −1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getgrnam_r*().

```
long int initlen = sysconf(_SC_GETGR_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct group result;
struct group *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getgrnam_r("somegroup", &result, buffer, len, &resultp))
        == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

APPLICATION USAGE

The *getgrnam_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return −1 from *sysconf*() indicating that there is no maximum for _SC_GETGR_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endgrent( ), *getgrgid*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<grp.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getgroups — get supplementary group IDs

SYNOPSIS

#include <unistd.h>

int getgroups(int *gidsetsize*, gid_t *grouplist*[]);

DESCRIPTION

The *getgroups*() function shall fill in the array *grouplist* with the current supplementary group IDs of the calling process. It is implementation-defined whether *getgroups*() also returns the effective group ID in the *grouplist* array.

The *gidsetsize* argument specifies the number of elements in the array *grouplist*. The actual number of group IDs stored in the array shall be returned. The values of array entries with indices greater than or equal to the value returned are undefined.

If *gidsetsize* is 0, *getgroups*() shall return the number of group IDs that it would otherwise return without modifying the array pointed to by *grouplist*.

If the effective group ID of the process is returned with the supplementary group IDs, the value returned shall always be greater than or equal to one and less than or equal to the value of {NGROUPS_MAX}+1.

RETURN VALUE

Upon successful completion, the number of supplementary group IDs shall be returned. A return value of −1 indicates failure and *errno* shall be set to indicate the error.

ERRORS

The *getgroups*() function shall fail if:

EINVAL
The *gidsetsize* argument is non-zero and less than the number of group IDs that would have been returned.

The following sections are informative.

EXAMPLES

Getting the Supplementary Group IDs of the Calling Process

The following example places the current supplementary group IDs of the calling process into the *group* array.

```
#include <sys/types.h>
#include <unistd.h>
...
gid_t *group;
int nogroups;
long ngroups_max;

ngroups_max = sysconf(_SC_NGROUPS_MAX) + 1;
group = (gid_t *)malloc(ngroups_max *sizeof(gid_t));

ngroups = getgroups(ngroups_max, group);
```

APPLICATION USAGE

None.

RATIONALE

The related function *setgroups*() is a privileged operation and therefore is not covered by this volume of POSIX.1-2017.

As implied by the definition of supplementary groups, the effective group ID may appear in the array returned by *getgroups*() or it may be returned only by *getegid*(). Duplication may exist, but the application needs to call *getegid*() to be sure of getting all of the information. Various implementation variations and administrative sequences cause the set of groups appearing in the result of *getgroups*() to vary in order and as to whether the effective group ID is included, even when the set of groups is the same (in the mathematical sense of ''set''). (The history of a process and its parents could affect the details of the result.)

Application developers should note that {NGROUPS_MAX} is not necessarily a constant on all implementations.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *setgid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gethostent — network host database functions

SYNOPSIS

```
#include <netdb.h>

struct hostent *gethostent(void);
```

DESCRIPTION

Refer to *endhostent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gethostid — get an identifier for the current host

SYNOPSIS

```
#include <unistd.h>

long gethostid(void);
```

DESCRIPTION

The *gethostid*() function shall retrieve a 32-bit identifier for the current host.

RETURN VALUE

Upon successful completion, *gethostid*() shall return an identifier for the current host.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

This volume of POSIX.1-2017 does not define the domain in which the return value is unique.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

initstate( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gethostname — get name of current host

SYNOPSIS

#include <unistd.h>

int gethostname(char **name*, size_t *namelen*);

DESCRIPTION

The *gethostname*() function shall return the standard host name for the current machine. The *namelen* argument shall specify the size of the array pointed to by the *name* argument. The returned name shall be null-terminated, except that if *namelen* is an insufficient length to hold the host name, then the returned name shall be truncated and it is unspecified whether the returned name is null-terminated.

Host names are limited to {HOST_NAME_MAX} bytes.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

gethostid( ), *uname*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getitimer, setitimer — get and set value of interval timer

SYNOPSIS

```
#include <sys/time.h>

int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *restrict value,
    struct itimerval *restrict ovalue);
```

DESCRIPTION

The *getitimer*() function shall store the current value of the timer specified by *which* into the structure pointed to by *value*. The *setitimer*() function shall set the timer specified by *which* to the value specified in the structure pointed to by *value*, and if *ovalue* is not a null pointer, store the previous value of the timer in the structure pointed to by *ovalue*.

A timer value is defined by the **itimerval** structure, specified in *<sys/time.h>*. If *it_value* is non-zero, it shall indicate the time to the next timer expiration. If *it_interval* is non-zero, it shall specify a value to be used in reloading *it_value* when the timer expires. Setting *it_value* to 0 shall disable a timer, regardless of the value of *it_interval*. Setting *it_interval* to 0 shall disable a timer after its next expiration (assuming *it_value* is non-zero).

Implementations may place limitations on the granularity of timer values. For each interval timer, if the requested timer value requires a finer granularity than the implementation supports, the actual timer value shall be rounded up to the next supported value.

An XSI-conforming implementation provides each process with at least three interval timers, which are indicated by the *which* argument:

ITIMER_PROF Decrements both in process virtual time and when the system is running on behalf of the process. It is designed to be used by interpreters in statistically profiling the execution of interpreted programs. Each time the ITIMER_PROF timer expires, the SIGPROF signal is delivered.

ITIMER_REAL
Decrements in real time. A SIGALRM signal is delivered when this timer expires.

ITIMER_VIRTUAL
Decrements in process virtual time. It runs only when the process is executing. A SIGVTALRM signal is delivered when it expires.

The interaction between *setitimer*() and *alarm*() or *sleep*() is unspecified.

RETURN VALUE

Upon successful completion, *getitimer*() or *setitimer*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setitimer*() function shall fail if:

EINVAL
The *value* argument is not in canonical form. (In canonical form, the number of microseconds is a non-negative integer less than 1 000 000 and the number of seconds is a non-negative integer.)

The *getitimer*() and *setitimer*() functions may fail if:

EINVAL
The *which* argument is not recognized.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications should use the *timer_gettime*() and *timer_settime*() functions instead of the obsolescent *getitimer*() and *setitimer*() functions, respectively.

RATIONALE

None.

FUTURE DIRECTIONS

The *getitimer*() and *setitimer*() functions may be removed in a future version.

SEE ALSO

alarm( ), *exec*, *sleep*( ), *timer_getoverrun*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<sys_time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getline — read a delimited record from *stream*

SYNOPSIS

```
#include <stdio.h>

ssize_t getline(char **restrict lineptr, size_t *restrict n,
   FILE *restrict stream);
```

DESCRIPTION

Refer to *getdelim*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getlogin, getlogin_r — get login name

SYNOPSIS

```
#include <unistd.h>

char *getlogin(void);
int getlogin_r(char *name, size_t namesize);
```

DESCRIPTION

The *getlogin*() function shall return a pointer to a string containing the user name associated by the login activity with the controlling terminal of the current process. If *getlogin*() returns a non-null pointer, then that pointer points to the name that the user logged in under, even if there are several login names with the same user ID.

The *getlogin*() function need not be thread-safe.

The *getlogin_r*() function shall put the name associated by the login activity with the controlling terminal of the current process in the character array pointed to by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum size of the login name is {LOGIN_NAME_MAX}.

If *getlogin_r*() is successful, *name* points to the name the user used at login, even if there are several login names with the same user ID.

The *getlogin*() and *getlogin_r*() functions may make use of file descriptors 0, 1, and 2 to find the controlling terminal of the current process, examining each in turn until the terminal is found. If in this case none of these three file descriptors is open to the controlling terminal, these functions may fail. The method used to find the terminal associated with a file descriptor may depend on the file descriptor being open to the actual terminal device, not **/dev/tty**.

RETURN VALUE

Upon successful completion, *getlogin*() shall return a pointer to the login name or a null pointer if the user's login name cannot be found. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *getlogin*(). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

If successful, the *getlogin_r*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

ENOTTY
None of the file descriptors 0, 1, or 2 is open to the controlling terminal of the current process.

ENXIO
The calling process has no controlling terminal.

The *getlogin_r*() function may fail if:

ERANGE
The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

The following sections are informative.

EXAMPLES

Getting the User Login Name S

The following example calls the *getlogin*() function to obtain the name of the user associated with the calling process, and passes this information to the *getpwnam*() function to get the associated user database information.

```
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
    }
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*()) shall return the name associated with the effective user ID of the process; *getlogin*() shall return the name associated with the current login activity; and *getpwuid*(*getuid*()) shall return the name associated with the real user ID of the process.

The *getlogin_r*() function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The *getlogin*() function returns a pointer to the user's login name. The same user ID may be shared by several login names. If it is desired to get the user database entry that is used during login, the result of *getlogin*() should be used to provide the argument to the *getpwnam*() function. (This might be used to determine the user's login shell, particularly where a single user has multiple login shells with distinct login names, but the same user ID.)

The information provided by the *cuserid*( ) function, which was originally defined in the POSIX.1-1988 standard and subsequently removed, can be obtained by the following:

```
getpwuid(geteuid())
```

while the information provided by historical implementations of *cuserid*( ) can be obtained by:

```
getpwuid(getuid())
```

The thread-safe version of this function places the user name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam( ), *getpwuid*( ), *geteuid*( ), *getuid*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getmsg, getpmsg — receive next message from a STREAMS file (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int getmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict flagsp);
int getpmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict bandp,
    int *restrict flagsp);
```

DESCRIPTION

The *getmsg*() function shall retrieve the contents of a message located at the head of the STREAM head read queue associated with a STREAMS file and place the contents into one or more buffers. The message contains either a data part, a control part, or both. The data and control parts of the message shall be placed into separate buffers, as described below. The semantics of each part are defined by the originator of the message.

The *getpmsg*() function shall be equivalent to *getmsg*(), except that it provides finer control over the priority of the messages received. Except where noted, all requirements on *getmsg*() also pertain to *getpmsg*().

The *fildes* argument specifies a file descriptor referencing a STREAMS-based file.

The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure, in which the *buf* member points to a buffer in which the data or control information is to be placed, and the *maxlen* member indicates the maximum number of bytes this buffer can hold. On return, the *len* member shall contain the number of bytes of data or control information actually received. The *len* member shall be set to 0 if there is a zero-length control or data part and *len* shall be set to −1 if no data or control information is present in the message.

When *getmsg*() is called, *flagsp* should point to an integer that indicates the type of message the process is able to receive. This is described further below.

The *ctlptr* argument is used to hold the control part of the message, and *dataptr* is used to hold the data part of the message. If *ctlptr* (or *dataptr*) is a null pointer or the *maxlen* member is −1, the control (or data) part of the message shall not be processed and shall be left on the STREAM head read queue, and if the *ctlptr* (or *dataptr*) is not a null pointer, *len* shall be set to −1. If the *maxlen* member is set to 0 and there is a zero-length control (or data) part, that zero-length part shall be removed from the read queue and *len* shall be set to 0. If the *maxlen* member is set to 0 and there are more than 0 bytes of control (or data) information, that information shall be left on the read queue and *len* shall be set to 0. If the *maxlen* member in *ctlptr* (or *dataptr*) is less than the control (or data) part of the message, *maxlen* bytes shall be retrieved. In this case, the remainder of the message shall be left on the STREAM head read queue and a non-zero return value shall be provided.

By default, *getmsg*() shall process the first available message on the STREAM head read queue. However, a process may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to RS_HIPRI. In this case, *getmsg*() shall only process the next message if it is a high-priority message. When the integer pointed to by *flagsp* is 0, any available message shall be retrieved. In this case, on return, the integer pointed to by *flagsp* shall be set to RS_HIPRI if a high-priority message was retrieved, or 0 otherwise.

For *getpmsg*(), the flags are different. The *flagsp* argument points to a bitmask with the following mutually-exclusive flags defined: MSG_HIPRI, MSG_BAND, and MSG_ANY. Like *getmsg*(), *getpmsg*() shall process the first available message on the STREAM head read queue. A process may choose to retrieve only high-priority messages by setting the integer pointed to by *flagsp* to MSG_HIPRI and the integer pointed to by *bandp* to 0. In this case, *getpmsg*() shall only process the next message if it is a high-priority message. In a similar manner, a process may choose to retrieve a message from a particular priority band by setting the integer pointed to by *flagsp* to MSG_BAND and the integer pointed to by *bandp* to the priority band of interest. In this case, *getpmsg*() shall only process the next message if it is in a priority band equal to, or greater than, the integer pointed to by *bandp*, or if it is a high-priority message. If a process wants to get the first message off the queue, the integer pointed to by *flagsp* should be set to MSG_ANY and the integer pointed to by *bandp* should be set to 0. On return, if the message retrieved was a high-priority message, the integer pointed to by *flagsp* shall be set to MSG_HIPRI and the integer pointed to by *bandp* shall be set to 0. Otherwise, the integer pointed to by *flagsp* shall be set to MSG_BAND and the integer pointed to by *bandp* shall be set to the priority band of the message.

If O_NONBLOCK is not set, *getmsg*() and *getpmsg*() shall block until a message of the type specified by *flagsp* is available at the front of the STREAM head read queue. If O_NONBLOCK is set and a message of the specified type is not present at the front of the read queue, *getmsg*() and *getpmsg*() shall fail and set *errno* to **[EAGAIN]**.

If a hangup occurs on the STREAM from which messages are retrieved, *getmsg*() and *getpmsg*() shall continue to operate normally, as described above, until the STREAM head read queue is empty. Thereafter, they shall return 0 in the *len* members of *ctlptr* and *dataptr*.

RETURN VALUE

Upon successful completion, *getmsg*() and *getpmsg*() shall return a non-negative value. A value of 0 indicates that a full message was read successfully. A return value of MORECTL indicates that more control information is waiting for retrieval. A return value of MOREDATA indicates that more data is waiting for retrieval. A return value of the bitwise-logical OR of MORECTL and MOREDATA indicates that both types of information remain. Subsequent *getmsg*() and *getpmsg*() calls shall retrieve the remainder of the message. However, if a message of higher priority has come in on the STREAM head read queue, the next call to *getmsg*() or *getpmsg*() shall retrieve that higher-priority message before retrieving the remainder of the previous message.

If the high priority control part of the message is consumed, the message shall be placed back on the queue as a normal message of band 0. Subsequent *getmsg*() and *getpmsg*() calls shall retrieve the remainder of the message. If, however, a priority message arrives or already exists on the STREAM head, the subsequent call to *getmsg*() or *getpmsg*() shall retrieve the higher-priority message before retrieving the remainder of the message that was put back.

Upon failure, *getmsg*() and *getpmsg*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *getmsg*() and *getpmsg*() functions shall fail if:

EAGAIN
The O_NONBLOCK flag is set and no messages are available.

EBADF
The *fildes* argument is not a valid file descriptor open for reading.

EBADMSG
The queued message to be read is not valid for *getmsg*() or *getpmsg*() or a pending file descriptor is at the STREAM head.

EINTR
A signal was caught during *getmsg*() or *getpmsg*().

EINVAL
An illegal value was specified by *flagsp*, or the STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

ENOSTR
A STREAM is not associated with *fildes*.

In addition, *getmsg*() and *getpmsg*() shall fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of *getmsg*() or *getpmsg*() but reflects the prior error.

The following sections are informative.

EXAMPLES

Getting Any Message

In the following example, the value of *fd* is assumed to refer to an open STREAMS file. The call to *getmsg*() retrieves any available message on the associated STREAM-head read queue, returning control and data information to the buffers pointed to by *ctrlbuf* and *databuf*, respectively.

```
#include <stropts.h>
...
int fd;
char ctrlbuf[128];
char databuf[512];
struct strbuf ctrl;
struct strbuf data;
int flags = 0;
int ret;

ctrl.buf = ctrlbuf;
ctrl.maxlen = sizeof(ctrlbuf);

data.buf = databuf;
data.maxlen = sizeof(databuf);

ret = getmsg (fd, &ctrl, &data, &flags);
```

Getting the First Message off the Queue

In the following example, the call to *getpmsg*() retrieves the first available message on the associated STREAM-head read queue.

```
#include <stropts.h>
...

int fd;
char ctrlbuf[128];
char databuf[512];
struct strbuf ctrl;
struct strbuf data;
int band = 0;
int flags = MSG_ANY;
int ret;

ctrl.buf = ctrlbuf;
ctrl.maxlen = sizeof(ctrlbuf);

data.buf = databuf;
data.maxlen = sizeof(databuf);

ret = getpmsg (fd, &ctrl, &data, &band, &flags);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *getmsg*() and *getpmsg*() functions may be removed in a future version.

SEE ALSO

Section 2.6, *STREAMS*, *poll*( ), *putmsg*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getnameinfo — get name information

SYNOPSIS

```
#include <sys/socket.h>
#include <netdb.h>

int getnameinfo(const struct sockaddr *restrict sa, socklen_t salen,
   char *restrict node, socklen_t nodelen, char *restrict service,
   socklen_t servicelen, int flags);
```

DESCRIPTION

The *getnameinfo*() function shall translate a socket address to a node name and service location, all of which are defined as in *freeaddrinfo*( ).

The *sa* argument points to a socket address structure to be translated. The *salen* argument contains the length of the address pointed to by *sa*.

If the socket address structure contains an IPv4-mapped IPv6 address or an IPv4-compatible IPv6 address, the implementation shall extract the embedded IPv4 address and lookup the node name for that IPv4 address.

If the address is the IPv6 unspecified address (**"::"**), a lookup shall not be performed and the behavior shall be the same as when the node's name cannot be located.

If the *node* argument is non-NULL and the *nodelen* argument is non-zero, then the *node* argument points to a buffer able to contain up to *nodelen* bytes that receives the node name as a null-terminated string. If the *node* argument is NULL or the *nodelen* argument is zero, the node name shall not be returned. If the node's name cannot be located, the numeric form of the address contained in the socket address structure pointed to by the *sa* argument is returned instead of its name.

If the *service* argument is non-NULL and the *servicelen* argument is non-zero, then the *service* argument points to a buffer able to contain up to *servicelen* bytes that receives the service name as a null-terminated string. If the *service* argument is NULL or the *servicelen* argument is zero, the service name shall not be returned. If the service's name cannot be located, the numeric form of the service address (for example, its port number) shall be returned instead of its name.

The *flags* argument is a flag that changes the default actions of the function. By default the fully-qualified domain name (FQDN) for the host shall be returned, but:

* If the flag bit NI_NOFQDN is set, only the node name portion of the FQDN shall be returned for local hosts.
* If the flag bit NI_NUMERICHOST is set, the numeric form of the address contained in the socket address structure pointed to by the *sa* argument shall be returned instead of its name.
* If the flag bit NI_NAMEREQD is set, an error shall be returned if the host's name cannot be located.
* If the flag bit NI_NUMERICSERV is set, the numeric form of the service address shall be returned (for example, its port number) instead of its name.
* If the flag bit NI_NUMERICSCOPE is set, the numeric form of the scope identifier shall be returned (for example, interface index) instead of its name. This flag shall be ignored if the *sa* argument is not an IPv6 address.
* If the flag bit NI_DGRAM is set, this indicates that the service is a datagram service (SOCK_DGRAM). The default behavior shall assume that the service is a stream service (SOCK_STREAM).

Notes:

1. The two NI_NUMERICxxx flags are required to support the **−n** flag that many commands provide.
2. The NI_DGRAM flag is required for the few AF_INET and AF_INET6 port numbers (for example, [512,514]) that represent different services for UDP and TCP.

The *getnameinfo*() function shall be thread-safe.

RETURN VALUE

A zero return value for *getnameinfo*() indicates successful completion; a non-zero return value indicates failure. The possible values for the failures are listed in the ERRORS section.

Upon successful completion, *getnameinfo*() shall return the *node* and *service* names, if requested, in the buffers provided. The returned names are always null-terminated strings.

ERRORS

The *getnameinfo*() function shall fail and return the corresponding value if:

[EAI_AGAIN]
The name could not be resolved at this time. Future attempts may succeed.

[EAI_BADFLAGS]
The *flags* had an invalid value.

[EAI_FAIL] A non-recoverable error occurred.

[EAI_FAMILY]
The address family was not recognized or the address length was invalid for the specified family.

[EAI_MEMORY]
There was a memory allocation failure.

[EAI_NONAME]
The name does not resolve for the supplied parameters.

NI_NAMEREQD is set and the host's name cannot be located, or both *nodename* and *servname* were null.

[EAI_OVERFLOW]
An argument buffer overflowed. The buffer pointed to by the *node* argument or the *service* argument was too small.

[EAI_SYSTEM]
A system error occurred. The error code can be found in *errno*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the returned values are to be used as part of any further name resolution (for example, passed to *getaddrinfo*()), applications should provide buffers large enough to store any result possible on the system.

Given the IPv4-mapped IPv6 address **"::ffff:1.2.3.4"**, the implementation performs a lookup as if the socket address structure contains the IPv4 address **"1.2.3.4"**.

The IPv6 unspecified address (**"::"**) and the IPv6 loopback address (**"::1"**) are not IPv4-compatible addresses.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endservent( ), *freeaddrinfo*( ), *gai_strerror*( ), *inet_ntop*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<netdb.h>**, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getnetbyaddr, getnetbyname, getnetent — network database functions

SYNOPSIS

```
#include <netdb.h>

struct netent *getnetbyaddr(uint32_t net, int type);
struct netent *getnetbyname(const char *name);
struct netent *getnetent(void);
```

DESCRIPTION

Refer to *endnetent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getopt, optarg, opterr, optind, optopt — command option parsing

SYNOPSIS

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);
extern char *optarg;
extern int opterr, optind, optopt;
```

DESCRIPTION

The *getopt*() function is a command-line parser that shall follow Utility Syntax Guidelines 3, 4, 5, 6, 7, 9, and 10 in the Base Definitions volume of POSIX.1-2017, *Section 12.2*, *Utility Syntax Guidelines*.

The parameters *argc* and *argv* are the argument count and argument array as passed to *main*() (see *exec*()). The argument *optstring* is a string of recognized option characters; if a character is followed by a <colon>, the option takes an argument. All option characters allowed by Utility Syntax Guideline 3 are allowed in *optstring*. The implementation may accept other characters as an extension.

The variable *optind* is the index of the next element of the *argv*[ ] vector to be processed. It shall be initialized to 1 by the system, and *getopt*() shall update it when it finishes with each element of *argv*[ ]. If the application sets *optind* to zero before calling *getopt*(), the behavior is unspecified. When an element of *argv*[ ] contains multiple option characters, it is unspecified how *getopt*() determines which options have already been processed.

The *getopt*() function shall return the next option character (if one is found) from *argv* that matches a character in *optstring*, if there is one that matches. If the option takes an argument, *getopt*() shall set the variable *optarg* to point to the option-argument as follows:

1. If the option was the last character in the string pointed to by an element of *argv*, then *optarg* shall contain the next element of *argv*, and *optind* shall be incremented by 2. If the resulting value of *optind* is greater than *argc*, this indicates a missing option-argument, and *getopt*() shall return an error indication.
2. Otherwise, *optarg* shall point to the string following the option character in that element of *argv*, and *optind* shall be incremented by 1.

If, when *getopt*() is called:

argv[optind]	is a null pointer
**argv*[optind]	is not the character −
argv[optind]	points to the string "−"

getopt() shall return −1 without changing *optind*. If:

argv[optind]	points to the string "−−"

getopt() shall return −1 after incrementing *optind*.

If *getopt*() encounters an option character that is not contained in *optstring*, it shall return the <question-mark> (**'?'**) character. If it detects a missing option-argument, it shall return the <colon> character (**':'**) if the first character of *optstring* was a <colon>, or a <question-mark> character (**'?'**) otherwise. In either case, *getopt*() shall set the variable *optopt* to the option character that caused the error. If the application has not set the variable *opterr* to 0 and the first character of *optstring* is not a <colon>, *getopt*() shall also print a diagnostic message to *stderr* in the format specified for the *getopts* utility, unless the *stderr* stream has wide orientation, in which case the behavior is undefined.

The *getopt*() function need not be thread-safe.

RETURN VALUE

The *getopt*() function shall return the next option character specified on the command line.

A <colon> (**':'**) shall be returned if *getopt*() detects a missing argument and the first character of *optstring* was a <colon> (**':'**).

A <question-mark> (**'?'**) shall be returned if *getopt*() encounters an option character not in *optstring* or detects a missing argument and the first character of *optstring* was not a <colon> (**':'**).

Otherwise, *getopt*() shall return −1 when all command line options are parsed.

ERRORS

If the application has not set the variable *opterr* to 0, the first character of *optstring* is not a <colon>, and a write error occurs while *getopt*() is printing a diagnostic message to *stderr*, then the error indicator for *stderr* shall be set; but *getopt*() shall still succeed and the value of *errno* after *getopt*() is unspecified.

The following sections are informative.

EXAMPLES

Parsing Command Line Options

The following code fragment shows how you might process the arguments for a utility that can take the mutually-exclusive options *a* and *b* and the options *f* and *o*, both of which require arguments:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int
main(int argc, char *argv[ ])
{
    int c;
    int bflg = 0, aflg = 0, errflg = 0;
    char *ifile;
    char *ofile;
    . . .
    while ((c = getopt(argc, argv, ":abf:o:")) != -1) {
        switch(c) {
        case 'a':
            if (bflg)
                errflg++;
            else
                aflg++;
            break;
        case 'b':
            if (aflg)
                errflg++;
            else
                bflg++;
            break;
        case 'f':
            ifile = optarg;
            break;
        case 'o':
            ofile = optarg;
            break;
        case ':':       /* -f or -o without operand */
            fprintf(stderr,
                "Option -%c requires an operand\n", optopt);
            errflg++;
            break;
        case '?':
            fprintf(stderr,
                "Unrecognized option: '-%c'\n", optopt);
            errflg++;
        }
    }
    if (errflg) {
        fprintf(stderr, "usage: . . . ");
        exit(2);
    }
    for ( ; optind < argc; optind++) {
        if (access(argv[optind], R_OK)) {
    . . .
}
```

This code accepts any of the following as equivalent:

```
cmd -ao arg path path
cmd -a -o arg path path
cmd -o arg -a path path
cmd -a -o arg -- path path
cmd -a -oarg path path
cmd -aoarg path path
```

Selecting Options from the Command Line

The following example selects the type of database routines the user wants to use based on the *Options* argument.

```
#include <unistd.h>
#include <string.h>
...
const char *Options = "hdbtl";
...
int dbtype, c;
char *st;
...
dbtype = 0;
while ((c = getopt(argc, argv, Options)) != -1) {
    if ((st = strchr(Options, c)) != NULL) {
        dbtype = st - Options;
        break;
    }
}
```

APPLICATION USAGE

The *getopt*() function is only required to support option characters included in Utility Syntax Guideline 3. Many historical implementations of *getopt*() support other characters as options. This is an allowed extension, but applications that use extensions are not maximally portable. Note that support for multi-byte option characters is only possible when such characters can be represented as type **int**.

Applications which use wide-character output functions with *stderr* should ensure that any calls to *getopt*() do not write to *stderr*, either by setting *opterr* to 0 or by ensuring the first character of *optstring* is always a <colon>.

While *ferror*(*stderr*) may be used to detect failures to write a diagnostic to *stderr* when *getopt*() returns **'?'**, the value of *errno* is unspecified in such a condition. Applications desiring more control over handling write failures should set *opterr* to 0 and independently perform output to *stderr*, rather than relying on *getopt*() to do the output.

RATIONALE

The *optopt* variable represents historical practice and allows the application to obtain the identity of the invalid option.

The description has been written to make it clear that *getopt*(), like the *getopts* utility, deals with option-arguments whether separated from the option by <blank> characters or not. Note that the requirements on *getopt*() and *getopts* are more stringent than the Utility Syntax Guidelines.

The *getopt*() function shall return −1, rather than EOF, so that *<stdio.h>* is not required.

The special significance of a <colon> as the first character of *optstring* makes *getopt*() consistent with the *getopts* utility. It allows an application to make a distinction between a missing argument and an incorrect option letter without having to examine the option letter. It is true that a missing argument can only be detected in one case, but that is a case that has to be considered.

FUTURE DIRECTIONS

None.

SEE ALSO

exec

The Base Definitions volume of POSIX.1-2017, *Section 12.2*, *Utility Syntax Guidelines*, **<unistd.h>**

The Shell and Utilities volume of POSIX.1-2017, *getopts*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpeername — get the name of the peer socket

SYNOPSIS

```
#include <sys/socket.h>

int getpeername(int socket, struct sockaddr *restrict address,
   socklen_t *restrict address_len);
```

DESCRIPTION

The *getpeername*() function shall retrieve the peer address of the specified socket, store this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this address in the object pointed to by the *address_len* argument.

The *address_len* argument points to a **socklen_t** object which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address. If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the protocol permits connections by unbound clients, and the peer is not bound, then the value stored in the object pointed to by *address* is unspecified.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getpeername*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

EINVAL
The socket has been shut down.

ENOTCONN
The socket is not connected or otherwise has not had the peer pre-specified.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The operation is not supported for the socket protocol.

The *getpeername*() function may fail if:

ENOBUFS
Insufficient resources were available in the system to complete the call.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *getsockname*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpgid — get the process group ID for a process

SYNOPSIS

#include <unistd.h>

pid_t getpgid(pid_t *pid*);

DESCRIPTION

The *getpgid*() function shall return the process group ID of the process whose process ID is equal to *pid*. If *pid* is equal to 0, *getpgid*() shall return the process group ID of the calling process.

RETURN VALUE

Upon successful completion, *getpgid*() shall return a process group ID. Otherwise, it shall return (**pid_t**)−1 and set *errno* to indicate the error.

ERRORS

The *getpgid*() function shall fail if:

EPERM
The process whose process ID is equal to *pid* is not in the same session as the calling process, and the implementation does not allow access to the process group ID of that process from the calling process.

ESRCH
There is no process with a process ID equal to *pid*.

The *getpgid*() function may fail if:

EINVAL
The value of the *pid* argument is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpgrp*( ), *getpid*( ), *getsid*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpgrp — get the process group ID of the calling process

SYNOPSIS

```
#include <unistd.h>

pid_t getpgrp(void);
```

DESCRIPTION

The *getpgrp*() function shall return the process group ID of the calling process.

RETURN VALUE

The *getpgrp*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

4.3 BSD provides a *getpgrp*() function that returns the process group ID for a specified process. Although this function supports job control, all known job control shells always specify the calling process with this function. Thus, the simpler System V *getpgrp*() suffices, and the added complexity of the 4.3 BSD *getpgrp*() is provided by the XSI extension *getpgid*().

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpgid*( ), *getpid*( ), *getppid*( ), *kill*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpid — get the process ID

SYNOPSIS

```
#include <unistd.h>

pid_t getpid(void);
```

DESCRIPTION

The *getpid*() function shall return the process ID of the calling process.

RETURN VALUE

The *getpid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpgrp*( ), *getppid*( ), *kill*( ), *mkdtemp*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpmsg — receive next message from a STREAMS file

SYNOPSIS

```
#include <stropts.h>

int getpmsg(int fildes, struct strbuf *restrict ctlptr,
    struct strbuf *restrict dataptr, int *restrict bandp,
    int *restrict flagsp);
```

DESCRIPTION

Refer to *getmsg*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getppid — get the parent process ID

SYNOPSIS

```
#include <unistd.h>

pid_t getppid(void);
```

DESCRIPTION

The *getppid*() function shall return the parent process ID of the calling process.

RETURN VALUE

The *getppid*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpgid*( ), *getpgrp*( ), *getpid*( ), *kill*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpriority, setpriority — get and set the nice value

SYNOPSIS

```
#include <sys/resource.h>

int getpriority(int which, id_t who);
int setpriority(int which, id_t who, int value);
```

DESCRIPTION

The *getpriority*() function shall obtain the nice value of a process, process group, or user. The *setpriority*() function shall set the nice value of a process, process group, or user to *value*+{NZERO}.

Target processes are specified by the values of the *which* and *who* arguments. The *which* argument may be one of the following values: PRIO_PROCESS, PRIO_PGRP, or PRIO_USER, indicating that the *who* argument is to be interpreted as a process ID, a process group ID, or an effective user ID, respectively. A 0 value for the *who* argument specifies the current process, process group, or user.

The nice value set with *setpriority*() shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

If more than one process is specified, *getpriority*() shall return value {NZERO} less than the lowest nice value pertaining to any of the specified processes, and *setpriority*() shall set the nice values of all of the specified processes to *value*+{NZERO}.

The default nice value is {NZERO}; lower nice values shall cause more favorable scheduling. While the range of valid nice values is [0,{NZERO}*2−1], implementations may enforce more restrictive limits. If *value*+{NZERO} is less than the system's lowest supported nice value, *setpriority*() shall set the nice value to the lowest supported value; if *value*+{NZERO} is greater than the system's highest supported nice value, *setpriority*() shall set the nice value to the highest supported value.

Only a process with appropriate privileges can lower its nice value.

Any processes or threads using SCHED_FIFO or SCHED_RR shall be unaffected by a call to *setpriority*(). This is not considered an error. A process which subsequently reverts to SCHED_OTHER need not have its priority affected by such a *setpriority*() call.

The effect of changing the nice value may vary depending on the process-scheduling algorithm in effect.

Since *getpriority*() can return the value −1 upon successful completion, it is necessary to set *errno* to 0 prior to a call to *getpriority*(). If *getpriority*() returns the value −1, then *errno* can be checked to see if an error occurred or if the value is a legitimate nice value.

RETURN VALUE

Upon successful completion, *getpriority*() shall return an integer in the range −{NZERO} to {NZERO}−1. Otherwise, −1 shall be returned and *errno* set to indicate the error.

Upon successful completion, *setpriority*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getpriority*() and *setpriority*() functions shall fail if:

ESRCH
No process could be located using the *which* and *who* argument values specified.

EINVAL
The value of the *which* argument was not recognized, or the value of the *who* argument is not a valid process ID, process group ID, or user ID.

In addition, *setpriority*() may fail if:

EPERM
A process was located, but neither the real nor effective user ID of the executing process match the effective user ID of the process whose nice value is being changed.

EACCES
A request was made to change the nice value to a lower numeric value and the current process does not have appropriate privileges.

The following sections are informative.

EXAMPLES

Using getpriority( )

The following example returns the current scheduling priority for the process ID returned by the call to *getpid*().

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int ret;

pid = getpid();
ret = getpriority(which, pid);
```

Using setpriority( )

The following example sets the priority for the current process ID to −20.

```
#include <sys/resource.h>
...
int which = PRIO_PROCESS;
id_t pid;
int priority = -20;
int ret;

pid = getpid();
ret = setpriority(which, pid, priority);
```

APPLICATION USAGE

The *getpriority*() and *setpriority*() functions work with an offset nice value (nice value −{NZERO}). The nice value is in the range [0,2*{NZERO} −1], while the return value for *getpriority*() and the third parameter for *setpriority*() are in the range [−{NZERO},{NZERO} −1].

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

nice( ), *sched_get_priority_max*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sys_resource.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getprotobyname, getprotobynumber, getprotoent — network protocol database functions

SYNOPSIS

```
#include <netdb.h>

struct protoent *getprotobyname(const char *name);
struct protoent *getprotobynumber(int proto);
struct protoent *getprotoent(void);
```

DESCRIPTION

Refer to *endprotoent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpwent — get user database entry

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwent(void);
```

DESCRIPTION

Refer to *endpwent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpwnam, getpwnam_r — search user database for a name

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwnam(const char *name);
int getpwnam_r(const char *name, struct passwd *pwd, char *buffer,
   size_t bufsize, struct passwd **result);
```

DESCRIPTION

The *getpwnam*() function shall search the user database for an entry with a matching *name*.

The *getpwnam*() function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getpwnam*(). If *getpwnam*() returns a null pointer and *errno* is non-zero, an error occurred.

The *getpwnam_r*() function shall update the **passwd** structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the user database with a matching *name*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf*(_SC_GETPW_R_SIZE_MAX) returns either −1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The *getpwnam*() function shall return a pointer to a **struct passwd** with the structure as defined in <*pwd.h*> with a matching entry if found. A null pointer shall be returned if the requested entry is not found, or an error occurs. If the requested entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getpwent*(), *getpwnam*(), or *getpwuid*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

The *getpwnam_r*() function shall return zero on success or if the requested entry was not found and no error has occurred. If an error has occurred, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EIO An I/O error has occurred.

EINTR
A signal was caught during *getpwnam*().

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The *getpwnam_r*() function may fail if:

ERANGE
Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **passwd** structure.

The following sections are informative.

EXAMPLES

Note that *sysconf*(_SC_GETPW_R_SIZE_MAX) may return −1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwnam_r*().

```
long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwnam_r("someuser", &result, buffer, len, &resultp))
        == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Getting an Entry for the Login Name

The following example uses the *getlogin*() function to return the name of the user who logged in; this information is passed to the *getpwnam*() function to get the user database entry for that user.

```
#include <sys/types.h>
#include <pwd.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
...
char *lgn;
struct passwd *pw;
...
if ((lgn = getlogin()) == NULL || (pw = getpwnam(lgn)) == NULL) {
    fprintf(stderr, "Get of user information failed.\n"); exit(1);
}
...
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*()) returns the name associated with the effective user ID of the process; *getlogin*() returns the name associated with the current login activity; and *getpwuid*(*getuid*()) returns the name associated with the real user ID of the process.

The *getpwnam_r*() function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return −1 from *sysconf*() indicating that there is no maximum for _SC_GETPW_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwuid( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<pwd.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getpwuid, getpwuid_r — search user database for a user ID

SYNOPSIS

```
#include <pwd.h>

struct passwd *getpwuid(uid_t uid);
int getpwuid_r(uid_t uid, struct passwd *pwd, char *buffer,
   size_t bufsize, struct passwd **result);
```

DESCRIPTION

The *getpwuid*() function shall search the user database for an entry with a matching *uid*.

The *getpwuid*() function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *getpwuid*(). If *getpwuid*() returns a null pointer and *errno* is set to non-zero, an error occurred.

The *getpwuid_r*() function shall update the **passwd** structure pointed to by *pwd* and store a pointer to that structure at the location pointed to by *result*. The structure shall contain an entry from the user database with a matching *uid*. Storage referenced by the structure is allocated from the memory provided with the *buffer* parameter, which is *bufsize* bytes in size. A call to *sysconf*(_SC_GETPW_R_SIZE_MAX) returns either −1 without changing *errno* or an initial value suggested for the size of this buffer. A null pointer shall be returned at the location pointed to by *result* on error or if the requested entry is not found.

RETURN VALUE

The *getpwuid*() function shall return a pointer to a **struct passwd** with the structure as defined in <*pwd.h*> with a matching entry if found. A null pointer shall be returned if the requested entry is not found, or an error occurs. If the requested entry was not found, *errno* shall not be changed. On error, *errno* shall be set to indicate the error.

The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *getpwent*(), *getpwnam*(), or *getpwuid*(). The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

If successful, the *getpwuid_r*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EIO An I/O error has occurred.

EINTR
A signal was caught during *getpwuid*().

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The *getpwuid_r*() function may fail if:

ERANGE
Insufficient storage was supplied via *buffer* and *bufsize* to contain the data to be referenced by the resulting **passwd** structure.

The following sections are informative.

EXAMPLES

Note that *sysconf*(_SC_GETPW_R_SIZE_MAX) may return −1 if there is no hard limit on the size of the buffer needed to store all the groups returned. This example shows how an application can allocate a buffer of sufficient size to work with *getpwuid_r*().

```
long int initlen = sysconf(_SC_GETPW_R_SIZE_MAX);
size_t len;
if (initlen == -1)
    /* Default initial length. */
    len = 1024;
else
    len = (size_t) initlen;
struct passwd result;
struct passwd *resultp;
char *buffer = malloc(len);
if (buffer == NULL)
    ...handle error...
int e;
while ((e = getpwuid_r(42, &result, buffer, len, &resultp)) == ERANGE)
    {
    size_t newlen = 2 * len;
    if (newlen < len)
        ...handle error...
    len = newlen;
    char *newbuffer = realloc(buffer, len);
    if (newbuffer == NULL)
        ...handle error...
    buffer = newbuffer;
    }
if (e != 0)
    ...handle error...
free (buffer);
```

Getting an Entry for the Root User

The following example gets the user database entry for the user with user ID 0 (root).

```
#include <sys/types.h>
#include <pwd.h>
...
uid_t id = 0;
struct passwd *pwd;

pwd = getpwuid(id);
```

Finding the Name for the Effective User ID

The following example defines *pws* as a pointer to a structure of type **passwd**, which is used to store the structure pointer returned by the call to the *getpwuid*() function. The *geteuid*() function shall return the effective user ID of the calling process; this is used as the search criteria for the *getpwuid*() function. The call to *getpwuid*() shall return a pointer to the structure containing that user ID value.

```
#include <unistd.h>
#include <sys/types.h>
```

```
#include <pwd.h>
...
struct passwd *pws;
pws = getpwuid(geteuid());
```

Finding an Entry in the User Database

The following example uses *getpwuid*() to search the user database for a user ID that was previously stored in a **stat** structure, then prints out the user name if it is found. If the user is not found, the program prints the numeric value of the user ID for the entry.

```
#include <sys/types.h>
#include <pwd.h>
#include <stdio.h>
...
struct stat statbuf;
struct passwd *pwd;
...
if ((pwd = getpwuid(statbuf.st_uid)) != NULL)
    printf(" %-8.8s", pwd->pw_name);
else
    printf(" %-8d", statbuf.st_uid);
```

APPLICATION USAGE

Three names associated with the current process can be determined: *getpwuid*(*geteuid*()) returns the name associated with the effective user ID of the process; *getlogin*() returns the name associated with the current login activity; and *getpwuid*(*getuid*()) returns the name associated with the real user ID of the process.

The *getpwuid_r*() function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

Portable applications should take into account that it is usual for an implementation to return −1 from *sysconf*() indicating that there is no maximum for _SC_GETPW_R_SIZE_MAX.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpwnam( ), *geteuid*( ), *getuid*( ), *getlogin*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<pwd.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getrlimit, setrlimit — control maximum resource consumption

SYNOPSIS

```
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlp);
int setrlimit(int resource, const struct rlimit *rlp);
```

DESCRIPTION

The *getrlimit*() function shall get, and the *setrlimit*() function shall set, limits on the consumption of a variety of resources.

Each call to either *getrlimit*() or *setrlimit*() identifies a specific resource to be operated upon as well as a resource limit. A resource limit is represented by an **rlimit** structure. The *rlim_cur* member specifies the current or soft limit and the *rlim_max* member specifies the maximum or hard limit. Soft limits may be changed by a process to any value that is less than or equal to the hard limit. A process may (irreversibly) lower its hard limit to any value that is greater than or equal to the soft limit. Only a process with appropriate privileges can raise a hard limit. Both hard and soft limits can be changed in a single call to *setrlimit*() subject to the constraints described above.

The value RLIM_INFINITY, defined in *<sys/resource.h>*, shall be considered to be larger than any other limit value. If a call to *getrlimit*() returns RLIM_INFINITY for a resource, it means the implementation shall not enforce limits on that resource. Specifying RLIM_INFINITY as any resource limit value on a successful call to *setrlimit*() shall inhibit enforcement of that resource limit.

The following resources are defined:

RLIMIT_CORE
: This is the maximum size of a **core** file, in bytes, that may be created by a process. A limit of 0 shall prevent the creation of a **core** file. If this limit is exceeded, the writing of a **core** file shall terminate at this size.

RLIMIT_CPU
: This is the maximum amount of CPU time, in seconds, used by a process. If this limit is exceeded, SIGXCPU shall be generated for the process. If the process is catching or ignoring SIGXCPU, or all threads belonging to that process are blocking SIGXCPU, the behavior is unspecified.

RLIMIT_DATA
: This is the maximum size of a data segment of the process, in bytes. If this limit is exceeded, the *malloc*() function shall fail with *errno* set to **[ENOMEM]**.

RLIMIT_FSIZE
: This is the maximum size of a file, in bytes, that may be created by a process. If a write or truncate operation would cause this limit to be exceeded, SIGXFSZ shall be generated for the thread. If the thread is blocking, or the process is catching or ignoring SIGXFSZ, continued attempts to increase the size of a file from end-of-file to beyond the limit shall fail with *errno* set to **[EFBIG]**.

RLIMIT_NOFILE
: This is a number one greater than the maximum value that the system may assign to a newly-created descriptor. If this limit is exceeded, functions that allocate a file descriptor shall fail with *errno* set to **[EMFILE]**. This limit constrains the number of file descriptors that a process may allocate.

RLIMIT_STACK
: This is the maximum size of the initial thread's stack, in bytes. The implementation does not automatically grow the stack beyond this limit. If this limit is exceeded, SIGSEGV

shall be generated for the thread. If the thread is blocking SIGSEGV, or the process is ignoring or catching SIGSEGV and has not made arrangements to use an alternate stack, the disposition of SIGSEGV shall be set to SIG_DFL before it is generated.

RLIMIT_AS	This is the maximum size of total available memory of the process, in bytes. If this limit is exceeded, the *malloc*() and *mmap*() functions shall fail with *errno* set to **[ENOMEM]**. In addition, the automatic stack growth fails with the effects outlined above.

When using the *getrlimit*() function, if a resource limit can be represented correctly in an object of type **rlim_t**, then its representation is returned; otherwise, if the value of the resource limit is equal to that of the corresponding saved hard limit, the value returned shall be RLIM_SAVED_MAX; otherwise, the value returned shall be RLIM_SAVED_CUR.

When using the *setrlimit*() function, if the requested new limit is RLIM_INFINITY, the new limit shall be ''no limit''; otherwise, if the requested new limit is RLIM_SAVED_MAX, the new limit shall be the corresponding saved hard limit; otherwise, if the requested new limit is RLIM_SAVED_CUR, the new limit shall be the corresponding saved soft limit; otherwise, the new limit shall be the requested value. In addition, if the corresponding saved limit can be represented correctly in an object of type **rlim_t** then it shall be overwritten with the new limit.

The result of setting a limit to RLIM_SAVED_MAX or RLIM_SAVED_CUR is unspecified unless a previous call to *getrlimit*() returned that value as the soft or hard limit for the corresponding resource limit.

The determination of whether a limit can be correctly represented in an object of type **rlim_t** is implementation-defined. For example, some implementations permit a limit whose value is greater than RLIM_INFINITY and others do not.

The *exec* family of functions shall cause resource limits to be saved.

RETURN VALUE

Upon successful completion, *getrlimit*() and *setrlimit*() shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error.

ERRORS

The *getrlimit*() and *setrlimit*() functions shall fail if:

EINVAL
An invalid *resource* was specified; or in a *setrlimit*() call, the new *rlim_cur* exceeds the new *rlim_max*.

EPERM
The limit specified to *setrlimit*() would have raised the maximum limit value, and the calling process does not have appropriate privileges.

The *setrlimit*() function may fail if:

EINVAL
The limit specified cannot be lowered because current usage is already higher than the limit.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the value of {_POSIX_OPEN_MAX} from <*limits.h*>, unexpected behavior may occur.

If a process attempts to set the hard limit or soft limit for RLIMIT_NOFILE to less than the highest currently open file descriptor +1, unexpected behavior may occur.

RATIONALE

It should be noted that RLIMIT_STACK applies ''at least'' to the stack of the initial thread in the process, and not to the sum of all the stacks in the process, as that would be very limiting unless the value is so big as to provide no value at all with a single thread.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *malloc*( ), *open*( ), *sigaltstack*( ), *sysconf*( ), *ulimit*( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**, **<sys_resource.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getrusage — get information about resource utilization

SYNOPSIS

```
#include <sys/resource.h>

int getrusage(int who, struct rusage *r_usage);
```

DESCRIPTION

The *getrusage*() function shall provide measures of the resources used by the current process or its terminated and waited-for child processes. If the value of the *who* argument is RUSAGE_SELF, information shall be returned about resources used by the current process. If the value of the *who* argument is RUSAGE_CHILDREN, information shall be returned about resources used by the terminated and waited-for children of the current process. If the child is never waited for (for example, if the parent has SA_NOCLDWAIT set or sets SIGCHLD to SIG_IGN), the resource information for the child process is discarded and not included in the resource information provided by *getrusage*().

The *r_usage* argument is a pointer to an object of type **struct rusage** in which the returned information is stored.

RETURN VALUE

Upon successful completion, *getrusage*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getrusage*() function shall fail if:

EINVAL
The value of the *who* argument is not valid.

The following sections are informative.

EXAMPLES

Using getrusage( )

The following example returns information about the resources used by the current process.

```
#include <sys/resource.h>
...
int who = RUSAGE_SELF;
struct rusage usage;
int ret;

ret = getrusage(who, &usage);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *sigaction*( ), *time*( ), *times*( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, **<sys_resource.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gets — get a string from a *stdin* stream

SYNOPSIS

```
#include <stdio.h>

char *gets(char *s);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *gets*() function shall read bytes from the standard input stream, *stdin*, into the array pointed to by *s*, until a <newline> is read or an end-of-file condition is encountered. Any <newline> shall be discarded and a null byte shall be placed immediately after the last byte read into the array.

The *gets*() function may mark the last data access timestamp of the file associated with *stream* for update. The last data access timestamp shall be marked for update by the first successful execution of *fgetc*(), *fgets*(), *fread*(), *fscanf*(), *getc*(), *getchar*(), *getdelim*(), *getline*(), *gets*(), or *scanf*() using *stream* that returns data not supplied by a prior call to *ungetc*().

RETURN VALUE

Upon successful completion, *gets*() shall return *s*. If the end-of-file indicator for the stream is set, or if the stream is at end-of-file, the end-of-file indicator for the stream shall be set and *gets*() shall return a null pointer. If a read error occurs, the error indicator for the stream shall be set, *gets*() shall return a null pointer, and set *errno* to indicate the error.

ERRORS

Refer to *fgetc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Reading a line that overflows the array pointed to by *s* results in undefined behavior. The use of *fgets*() is recommended.

Since the user cannot specify the length of the buffer passed to *gets*(), use of this function is discouraged. The length of the string read is unlimited. It is possible to overflow this buffer in such a way as to cause applications to fail, or possible system security violations.

Applications should use the *fgets*() function instead of the obsolescent *gets*() function.

RATIONALE

The standard developers decided to mark the *gets*() function as obsolescent even though it is in the ISO C standard due to the possibility of buffer overflow.

FUTURE DIRECTIONS

The *gets*() function may be removed in a future version.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *feof*( ), *ferror*( ), *fgets*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getservbyname, getservbyport, getservent — network services database functions

SYNOPSIS

```
#include <netdb.h>

struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
struct servent *getservent(void);
```

DESCRIPTION

Refer to *endservent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getsid — get the process group ID of a session leader

SYNOPSIS

```
#include <unistd.h>

pid_t getsid(pid_t pid);
```

DESCRIPTION

The *getsid*() function shall obtain the process group ID of the process that is the session leader of the process specified by *pid*. If *pid* is (**pid_t**)0, it specifies the calling process.

RETURN VALUE

Upon successful completion, *getsid*() shall return the process group ID of the session leader of the specified process. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *getsid*() function shall fail if:

EPERM
The process specified by *pid* is not in the same session as the calling process, and the implementation does not allow access to the process group ID of the session leader of that process from the calling process.

ESRCH
There is no process with a process ID equal to *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fork*( ), *getpid*( ), *getpgid*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getsockname — get the socket name

SYNOPSIS

```
#include <sys/socket.h>

int getsockname(int socket, struct sockaddr *restrict address,
    socklen_t *restrict address_len);
```

DESCRIPTION

The *getsockname*() function shall retrieve the locally-bound name of the specified socket, store this address in the **sockaddr** structure pointed to by the *address* argument, and store the length of this address in the object pointed to by the *address_len* argument.

The *address_len* argument points to a **socklen_t** object which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address. If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the socket has not been bound to a local name, the value stored in the object pointed to by *address* is unspecified.

RETURN VALUE

Upon successful completion, 0 shall be returned, the *address* argument shall point to the address of the socket, and the *address_len* argument shall point to the length of the address. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getsockname*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The operation is not supported for this socket's protocol.

The *getsockname*() function may fail if:

EINVAL
The socket has been shut down.

ENOBUFS
Insufficient resources were available in the system to complete the function.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *bind*( ), *getpeername*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getsockopt — get the socket options

SYNOPSIS

```
#include <sys/socket.h>

int getsockopt(int socket, int level, int option_name,
   void *restrict option_value, socklen_t *restrict option_len);
```

DESCRIPTION

The *getsockopt*() function manipulates options associated with a socket.

The *getsockopt*() function shall retrieve the value for the option specified by the *option_name* argument for the socket specified by the *socket* argument. If the size of the option value is greater than *option_len*, the value stored in the object pointed to by the *option_value* argument shall be silently truncated. Otherwise, the object pointed to by the *option_len* argument shall be modified to indicate the actual length of the value.

The *level* argument specifies the protocol level at which the option resides. To retrieve options at the socket level, specify the *level* argument as SOL_SOCKET. To retrieve options at other levels, supply the appropriate level identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transmission Control Protocol), set *level* to IPPROTO_TCP as defined in the *<netinet/in.h>* header.

The socket in use may require the process to have appropriate privileges to use the *getsockopt*() function.

The *option_name* argument specifies a single option to be retrieved. It can be one of the socket-level options defined in **<sys_socket.h>** and described in *Section 2.10.16*, *Use of Options*.

RETURN VALUE

Upon successful completion, *getsockopt*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *getsockopt*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

EINVAL
The specified option is invalid at the specified socket level.

ENOPROTOOPT
The option is not supported by the protocol.

ENOTSOCK
The *socket* argument does not refer to a socket.

The *getsockopt*() function may fail if:

EACCES
The calling process does not have appropriate privileges.

EINVAL
The socket has been shut down.

ENOBUFS
Insufficient resources are available in the system to complete the function.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10.16, *Use of Options*, *bind*( ), *close*( ), *endprotoent*( ), *setsockopt*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**, **<netinet_in.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getsubopt — parse suboption arguments from a string

SYNOPSIS

#include <stdlib.h>

int getsubopt(char ***optionp*, char * const **keylistp*, char ***valuep*);

DESCRIPTION

The *getsubopt*() function shall parse suboption arguments in a flag argument. Such options often result from the use of *getopt*().

The *getsubopt*() argument *optionp* is a pointer to a pointer to the option argument string. The suboption arguments shall be separated by <comma> characters and each may consist of either a single token, or a token-value pair separated by an <equals-sign>.

The *keylistp* argument shall be a pointer to a vector of strings. The end of the vector is identified by a null pointer. Each entry in the vector is one of the possible tokens that might be found in **optionp*. Since <comma> characters delimit suboption arguments in *optionp*, they should not appear in any of the strings pointed to by *keylistp*. Similarly, because an <equals-sign> separates a token from its value, the application should not include an <equals-sign> in any of the strings pointed to by *keylistp*. The *getsubopt*() function shall not modify the *keylistp* vector.

The *valuep* argument is the address of a value string pointer.

If a <comma> appears in *optionp*, it shall be interpreted as a suboption separator. After <comma> characters have been processed, if there are one or more <equals-sign> characters in a suboption string, the first <equals-sign> in any suboption string shall be interpreted as a separator between a token and a value. Subsequent <equals-sign> characters in a suboption string shall be interpreted as part of the value.

If the string at **optionp* contains only one suboption argument (equivalently, no <comma> characters), *getsubopt*() shall update **optionp* to point to the null character at the end of the string. Otherwise, it shall isolate the suboption argument by replacing the <comma> separator with a null character, and shall update **optionp* to point to the start of the next suboption argument. If the suboption argument has an associated value (equivalently, contains an <equals-sign>), *getsubopt*() shall update **valuep* to point to the value's first character. Otherwise, it shall set **valuep* to a null pointer. The calling application may use this information to determine whether the presence or absence of a value for the suboption is an error.

Additionally, when *getsubopt*() fails to match the suboption argument with a token in the *keylistp* array, the calling application should decide if this is an error, or if the unrecognized option should be processed in another way.

RETURN VALUE

The *getsubopt*() function shall return the index of the matched token string, or −1 if no token strings were matched.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Parsing Suboptions

The following example uses the *getsubopt*() function to parse a *value* argument in the *optarg* external variable returned by a call to *getopt*().

```
#include <stdio.h>
```

```
#include <stdlib.h>
#include <unistd.h>

int do_all;
const char *type;
int read_size;
int write_size;
int read_only;

enum
{
    RO_OPTION = 0,
    RW_OPTION,
    READ_SIZE_OPTION,
    WRITE_SIZE_OPTION
};

const char *mount_opts[] =
{
    [RO_OPTION] = "ro",
    [RW_OPTION] = "rw",
    [READ_SIZE_OPTION] = "rsize",
    [WRITE_SIZE_OPTION] = "wsize",
    NULL
};

int
main(int argc, char *argv[])
{
    char *subopts, *value;
    int opt;

    while ((opt = getopt(argc, argv, "at:o:")) != -1)
        switch(opt)
            {
            case 'a':
                do_all = 1;
                break;
            case 't':
                type = optarg;
                break;
            case 'o':
                subopts = optarg;
                while (*subopts != ' ')
                {
                    char *saved = subopts;
                    switch(getsubopt(&subopts, (char **)mount_opts,
                        &value))
                    {
                    case RO_OPTION:
                        read_only = 1;
                        break;
                    case RW_OPTION:
                        read_only = 0;
                        break;
                    case READ_SIZE_OPTION:
                        if (value == NULL)
                            abort();
                        read_size = atoi(value);
                        break;
                    case WRITE_SIZE_OPTION:
                        if (value == NULL)
                            abort();
                        write_size = atoi(value);
                        break;
                    default:
                        /* Unknown suboption. */
                        printf("Unknown suboption `%s'\n", saved);
                        abort();
                    }
                }
                break;
            default:
                abort();
            }

    /* Do the real work. */

    return 0;
}
```

If the above example is invoked with:

```
program -o ro,rsize=512
```

then after option parsing, the variable *do_all* will be 0, *type* will be a null pointer, *read_size* will be 512, *write_size* will be 0, and *read_only* will be 1. If it is invoked with:

```
program -o oops
```

it will print:

```
"Unknown suboption `oops'"
```

before aborting.

APPLICATION USAGE

The value of **valuep* when *getsubopt*() returns −1 is unspecified. Historical implementations provide various incompatible extensions to allow an application to access the suboption text that was not found in the *keylistp* array.

RATIONALE

The *keylistp* argument of *getsubopt*() is typed as **char * const *** to match historical practice. However, the standard is clear that implementations will not modify either the array or the strings contained in the array, as if the argument had been typed **const char * const ***.

FUTURE DIRECTIONS

None.

SEE ALSO

getopt( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gettimeofday — get the date and time

SYNOPSIS

```
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
```

DESCRIPTION

The *gettimeofday*() function shall obtain the current time, expressed as seconds and microseconds since the Epoch, and store it in the **timeval** structure pointed to by *tp*. The resolution of the system clock is unspecified.

If *tzp* is not a null pointer, the behavior is unspecified.

RETURN VALUE

The *gettimeofday*() function shall return 0 and no value shall be reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications should use the *clock_gettime*() function instead of the obsolescent *gettimeofday*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *gettimeofday*() function may be removed in a future version.

SEE ALSO

clock_getres( ), *ctime*( )

The Base Definitions volume of POSIX.1-2017, **<sys_time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getuid — get a real user ID

SYNOPSIS

```
#include <unistd.h>

uid_t getuid(void);
```

DESCRIPTION

The *getuid*() function shall return the real user ID of the calling process. The *getuid*() function shall not modify *errno*.

RETURN VALUE

The *getuid*() function shall always be successful and no return value is reserved to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID of the calling process to the real user ID.

```
#include <unistd.h>
...
seteuid(getuid());
```

APPLICATION USAGE

None.

RATIONALE

In a conforming environment, *getuid*() will always succeed. It is possible for implementations to provide an extension where a process in a non-conforming environment will not be associated with a user or group ID. It is recommended that such implementations return (**uid_t**)−1 and set *errno* to indicate such an environment; doing so does not violate this standard, since such an environment is already an extension.

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getgid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer’s Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getutxent, getutxid, getutxline — get user accounting database entries

SYNOPSIS

```
#include <utmpx.h>

struct utmpx *getutxent(void);
struct utmpx *getutxid(const struct utmpx *id);
struct utmpx *getutxline(const struct utmpx *line);
```

DESCRIPTION

Refer to *endutxent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getwc — get a wide character from a stream

SYNOPSIS

#include <stdio.h>
#include <wchar.h>

wint_t getwc(FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *getwc*() function shall be equivalent to *fgetwc*(), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fgetwc*( ).

ERRORS

Refer to *fgetwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *getwc*() may treat incorrectly a *stream* argument with side-effects. In particular, *getwc*(**f*++) does not necessarily work as expected. Therefore, use of this function is not recommended; *fgetwc*() should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetwc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

getwchar — get a wide character from a *stdin* stream

SYNOPSIS

```
#include <wchar.h>

wint_t getwchar(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *getwchar*() function shall be equivalent to *getwc*(*stdin*).

RETURN VALUE

Refer to *fgetwc*( ).

ERRORS

Refer to *fgetwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the **wint_t** value returned by *getwchar*() is stored into a variable of type **wchar_t** and then compared against the **wint_t** macro WEOF, the result may be incorrect. Only the **wint_t** type is guaranteed to be able to represent any wide character and WEOF.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fgetwc*( ), *getwc*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

glob, globfree — generate pathnames matching a pattern

SYNOPSIS

```
#include <glob.h>

int glob(const char *restrict pattern, int flags,
   int(*errfunc)(const char *epath, int eerrno),
   glob_t *restrict pglob);
void globfree(glob_t *pglob);
```

DESCRIPTION

The *glob*() function is a pathname generator that shall implement the rules defined in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13*, *Pattern Matching Notation*, with optional support for rule 3 in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13.3*, *Patterns Used for Filename Expansion*.

The structure type **glob_t** is defined in *<glob.h>* and includes at least the following members:

center box tab(!); cB | cB | cB lw(1.25i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ size_t!gl_pathc!Count of paths matched by *pattern*. char **!gl_pathv!Pointer to a list of matched pathnames. size_t!gl_offs!T{ Slots to reserve at the beginning of *gl_pathv*. T}

The argument *pattern* is a pointer to a pathname pattern to be expanded. The *glob*() function shall match all accessible pathnames against this pattern and develop a list of all pathnames that match. In order to have access to a pathname, *glob*() requires search permission on every component of a path except the last, and read permission on each directory of any filename component of *pattern* that contains any of the following special characters: **'*'**, **'?'**, and **'['**.

The *glob*() function shall store the number of matched pathnames into *pglob*−>*gl_pathc* and a pointer to a list of pointers to pathnames into *pglob*−>*gl_pathv*. The pathnames shall be in sort order as defined by the current setting of the *LC_COLLATE* category; see the Base Definitions volume of POSIX.1-2017, *Section 7.3.2*, *LC_COLLATE*. The first pointer after the last pathname shall be a null pointer. If the pattern does not match any pathnames, the returned number of matched paths is set to 0, and the contents of *pglob*−>*gl_pathv* are implementation-defined.

It is the caller's responsibility to create the structure pointed to by *pglob*. The *glob*() function shall allocate other space as needed, including the memory pointed to by *gl_pathv*. The *globfree*() function shall free any space associated with *pglob* from a previous call to *glob*().

The *flags* argument is used to control the behavior of *glob*(). The value of *flags* is a bitwise-inclusive OR of zero or more of the following constants, which are defined in *<glob.h>*:

GLOB_APPEND
: Append pathnames generated to the ones from a previous call to *glob*().

GLOB_DOOFFS
: Make use of *pglob*−>*gl_offs*. If this flag is set, *pglob*−>*gl_offs* is used to specify how many null pointers to add to the beginning of *pglob*−>*gl_pathv*. In other words, *pglob*−>*gl_pathv* shall point to *pglob*−>*gl_offs* null pointers, followed by *pglob*−>*gl_pathc* pathname pointers, followed by a null pointer.

GLOB_ERR
: Cause *glob*() to return when it encounters a directory that it cannot open or read. Ordinarily, *glob*() continues to find matches.

GLOB_MARK
: Each pathname that is a directory that matches *pattern* shall have a <slash> appended.

GLOB_NOCHECK
: Supports rule 3 in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13.3*, *Patterns Used for Filename Expansion*. If *pattern* does not match any pathname, then *glob*() shall return a list consisting of only *pattern*, and the number of matched pathnames is 1.

GLOB_NOESCAPE
: Disable backslash escaping.

GLOB_NOSORT
: Ordinarily, *glob*() sorts the matching pathnames according to the current setting of the *LC_COLLATE* category; see the Base Definitions volume of POSIX.1-2017, *Section 7.3.2*, *LC_COLLATE*. When this flag is used, the order of pathnames returned is unspecified.

The GLOB_APPEND flag can be used to append a new set of pathnames to those found in a previous call to *glob*(). The following rules apply to applications when two or more calls to *glob*() are made with the same value of *pglob* and without intervening calls to *globfree*():

1. The first such call shall not set GLOB_APPEND. All subsequent calls shall set it.
2. All the calls shall set GLOB_DOOFFS, or all shall not set it.
3. After the second call, *pglob*−>*gl_pathv* points to a list containing the following:
   a. Zero or more null pointers, as specified by GLOB_DOOFFS and *pglob*−>*gl_offs*.
   b. Pointers to the pathnames that were in the *pglob*−>*gl_pathv* list before the call, in the same order as before.
   c. Pointers to the new pathnames generated by the second call, in the specified order.
4. The count returned in *pglob*−>*gl_pathc* shall be the total number of pathnames from the two calls.
5. The application can change any of the fields after a call to *glob*(). If it does, the application shall reset them to the original value before a subsequent call, using the same *pglob* value, to *globfree*() or *glob*() with the GLOB_APPEND flag.

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not a null pointer, *glob*() calls *(*()*errfunc) with two arguments:

1. The *epath* argument is a pointer to the path that failed.
2. The *eerrno* argument is the value of *errno* from the failure, as set by *opendir*(), *readdir*(), or *stat*(). (Other values may be used to report other errors not explicitly documented for those functions.)

If *(*()*errfunc) is called and returns non-zero, or if the GLOB_ERR flag is set in *flags*, *glob*() shall stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If GLOB_ERR is not set and either *errfunc* is a null pointer or *(*()*errfunc) returns 0, the error shall be ignored.

The *glob*() function shall not fail because of large files.

RETURN VALUE

Upon successful completion, *glob*() shall return 0. The argument *pglob*−>*gl_pathc* shall return the number of matched pathnames and the argument *pglob*−>*gl_pathv* shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if *pglob*−>*gl_pathc* is 0, the content of *pglob*−>*gl_pathv* is undefined.

The *globfree*() function shall not return a value.

If *glob*() terminates due to an error, it shall return one of the non-zero constants defined in *<glob.h>*. The arguments *pglob*−>*gl_pathc* and *pglob*−>*gl_pathv* are still set as defined above.

ERRORS

The *glob*() function shall fail and return the corresponding value if:

GLOB_ABORTED
: The scan was stopped because GLOB_ERR was set or *(*)*errfunc) returned non-zero.

GLOB_NOMATCH
: The pattern does not match any existing pathname, and GLOB_NOCHECK was not set in flags.

GLOB_NOSPACE
: An attempt to allocate memory failed.

The following sections are informative.

EXAMPLES

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with *execv*(), *execve*(), or *execvp*(). Suppose, for example, that an application wants to do the equivalent of:

```
ls -l *.c
```

but for some reason:

```
system("ls -l *.c")
```

is not acceptable. The application could obtain approximately the same result using the sequence:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp("ls", &globbuf.gl_pathv[0]);
```

Using the same example:

```
ls -l *.c *.h
```

could be approximately simulated using GLOB_APPEND as follows:

```
globbuf.gl_offs = 2;
glob("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
...
```

APPLICATION USAGE

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch*().

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob*() fails. This allows *glob*() to report partial results in the event of an error. However, if *gl_pathc* is 0, *gl_pathv* is unspecified even if *glob*() did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The *sh* utility might use this for option-arguments, for example.

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

Applications that need tilde and parameter expansion should use *wordexp*().

RATIONALE

It was claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
      * sizeof(char *));
(void) memcpy(new+n, pglob->gl_pathv,
      pglob->gl_pathc * sizeof(char *));
(void) memset(new, 0, n * sizeof(char *));
free(pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc*(). This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc*() in a large chunk and then carved up within *glob*(), or it might have been created using a different memory allocator. It is not the intent of the standard developers to specify or imply how the memory used by *glob*() is managed.

The GLOB_APPEND flag would be used when an application wants to expand several different patterns into a single list.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fdopendir*( ), *fnmatch*( ), *fstatat*( ), *readdir*( ), *Section 2.6*, *Word Expansions*

The Base Definitions volume of POSIX.1-2017, *Section 7.3.2*, *LC_COLLATE*, **<glob.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

gmtime, gmtime_r — convert a time value to a broken-down UTC time

SYNOPSIS

```
#include <time.h>

struct tm *gmtime(const time_t *timer);
struct tm *gmtime_r(const time_t *restrict timer,
    struct tm *restrict result);
```

DESCRIPTION

For *gmtime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *gmtime*() function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as Coordinated Universal Time (UTC).

The relationship between a time in seconds since the Epoch used as an argument to *gmtime*() and the **tm** structure (defined in the *<time.h>* header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*), where the names in the structure and in the expression correspond.

The same relationship shall apply for *gmtime_r*().

The *gmtime*() function need not be thread-safe.

The *asctime*(), *ctime*(), *gmtime*(), and *localtime*() functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *gmtime_r*() function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time expressed as Coordinated Universal Time (UTC). The broken-down time is stored in the structure referred to by *result*. The *gmtime_r*() function shall also return the address of the same structure.

RETURN VALUE

Upon successful completion, the *gmtime*() function shall return a pointer to a **struct tm**. If an error is detected, *gmtime*() shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, *gmtime_r*() shall return the address of the structure pointed to by the argument *result*. If an error is detected, *gmtime_r*() shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *gmtime*() and *gmtime_r*() functions shall fail if:

EOVERFLOW
The result cannot be represented.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *gmtime_r*() function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

grantpt — grant access to the slave pseudo-terminal device

SYNOPSIS

```
#include <stdlib.h>

int grantpt(int fildes);
```

DESCRIPTION

The *grantpt*() function shall change the mode and ownership of the slave pseudo-terminal device associated with its master pseudo-terminal counterpart. The *fildes* argument is a file descriptor that refers to a master pseudo-terminal device. The user ID of the slave shall be set to the real UID of the calling process and the group ID shall be set to an unspecified group ID. The permission mode of the slave pseudo-terminal shall be set to readable and writable by the owner, and writable by the group.

The behavior of the *grantpt*() function is unspecified if the application has installed a signal handler to catch SIGCHLD signals.

RETURN VALUE

Upon successful completion, *grantpt*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *grantpt*() function may fail if:

EACCES
The corresponding slave pseudo-terminal device could not be accessed.

EBADF
The *fildes* argument is not a valid open file descriptor.

EINVAL
The *fildes* argument is not associated with a master pseudo-terminal device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See the RATIONALE section for *posix_openpt*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

open( ), *posix_openpt*( ), *ptsname*( ), *unlockpt*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

hcreate, hdestroy, hsearch — manage hash search table

SYNOPSIS

```
#include <search.h>

int hcreate(size_t nel);
void hdestroy(void);
ENTRY *hsearch(ENTRY item, ACTION action);
```

DESCRIPTION

The *hcreate*(), *hdestroy*(), and *hsearch*() functions shall manage hash search tables.

The *hcreate*() function shall allocate sufficient space for the table, and the application shall ensure it is called before *hsearch*() is used. The *nel* argument is an estimate of the maximum number of entries that the table shall contain. This number may be adjusted upward by the algorithm in order to obtain certain mathematically favorable circumstances.

The *hdestroy*() function shall dispose of the search table, and may be followed by another call to *hcreate*(). After the call to *hdestroy*(), the data can no longer be considered accessible.

The *hsearch*() function is a hash-table search routine. It shall return a pointer into a hash table indicating the location at which an entry can be found. The *item* argument is a structure of type **ENTRY** (defined in the *<search.h>* header) containing two pointers: *item.key* points to the comparison key (a **char ***), and *item.data* (a **void ***) points to any other data to be associated with that key. The comparison function used by *hsearch*() is *strcmp*(). The *action* argument is a member of an enumeration type **ACTION** indicating the disposition of the entry if it cannot be found in the table. ENTER indicates that the item should be inserted in the table at an appropriate point. FIND indicates that no entry should be made. Unsuccessful resolution is indicated by the return of a null pointer.

These functions need not be thread-safe.

RETURN VALUE

The *hcreate*() function shall return 0 if it cannot allocate sufficient space for the table; otherwise, it shall return non-zero.

The *hdestroy*() function shall not return a value.

The *hsearch*() function shall return a null pointer if either the action is FIND and the item could not be found or the action is ENTER and the table is full.

ERRORS

The *hcreate*() and *hsearch*() functions may fail if:

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

The following example reads in strings followed by two numbers and stores them in a hash table, discarding duplicates. It then reads in strings and finds the matching entry in the hash table and prints it out.

```
#include <stdio.h>
#include <search.h>
#include <string.h>

struct info {        /* This is the info stored in the table */
```

```
    int age, room;   /* other than the key. */
};

#define NUM_EMPL    5000    /* # of elements in search table. */

int main(void)
{
    char string_space[NUM_EMPL*20];   /* Space to store strings. */
    struct info info_space[NUM_EMPL]; /* Space to store employee info. */
    char *str_ptr = string_space;     /* Next space in string_space. */
    struct info *info_ptr = info_space;
                                /* Next space in info_space. */
    ENTRY item;
    ENTRY *found_item; /* Name to look for in table. */
    char name_to_find[30];

    int i = 0;

    /* Create table; no error checking is performed. */
    (void) hcreate(NUM_EMPL);
    while (scanf("%s%d%d", str_ptr, &info_ptr->age,
          &info_ptr->room) != EOF && i++ < NUM_EMPL) {

        /* Put information in structure, and structure in item. */
        item.key = str_ptr;
        item.data = info_ptr;
        str_ptr += strlen(str_ptr) + 1;
        info_ptr++;

        /* Put item into table. */
        (void) hsearch(item, ENTER);
    }

    /* Access table. */
    item.key = name_to_find;
    while (scanf("%s", item.key) != EOF) {
        if ((found_item = hsearch(item, FIND)) != NULL) {

            /* If item is in the table. */
            (void)printf("found %s, age = %d, room = %d\n",
                found_item->key,
                ((struct info *)found_item->data)->age,
                ((struct info *)found_item->data)->room);
        } else
            (void)printf("no such employee %s\n", name_to_find);
    }
    return 0;
}
```

APPLICATION USAGE

The *hcreate*() and *hsearch*() functions may use *malloc*() to allocate space.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

bsearch( ), *lsearch*( ), *malloc*( ), *strcmp*( ), *tdelete*( )

The Base Definitions volume of POSIX.1-2017, **<search.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

htonl, htons, ntohl, ntohs — convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t htonl(uint32_t hostlong);
uint16_t htons(uint16_t hostshort);
uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

These functions shall convert 16-bit and 32-bit quantities between network byte order and host byte order.

On some implementations, these functions are defined as macros.

The **uint32_t** and **uint16_t** types are defined in *<inttypes.h>*.

RETURN VALUE

The *htonl*() and *htons*() functions shall return the argument value converted from host to network byte order.

The *ntohl*() and *ntohs*() functions shall return the argument value converted from network to host byte order.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are most often used in conjunction with IPv4 addresses and ports as returned by *gethostent*() and *getservent*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endhostent( ), *endservent*( )

The Base Definitions volume of POSIX.1-2017, **<arpa_inet.h>**, **<inttypes.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

hypot, hypotf, hypotl — Euclidean distance function

SYNOPSIS

#include <math.h>

double hypot(double *x*, double *y*);
float hypotf(float *x*, float *y*);
long double hypotl(long double *x*, long double *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the value of the square root of x^2+y^2 without undue overflow or underflow.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the length of the hypotenuse of a right-angled triangle with sides of length *x* and *y*.

If the correct value would cause overflow, a range error shall occur and *hypot*(), *hypotf*(), and *hypotl*() shall return the value of the macro HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively.

If *x* or *y* is ±Inf, +Inf shall be returned (even if one of *x* or *y* is NaN).

If *x* or *y* is NaN, and the other is not ±Inf, a NaN shall be returned.

If both arguments are subnormal and the correct result is subnormal, a range error may occur and the correct result shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

See the EXAMPLES section in *atan2*().

APPLICATION USAGE

hypot(*x*,*y*), *hypot*(*y*,*x*), and *hypot*(*x*, −*y*) are equivalent.

hypot(*x*, ±0) is equivalent to *fabs*(*x*).

Underflow only happens when both *x* and *y* are subnormal and the (inexact) result is also subnormal.

These functions take precautions against overflow during intermediate steps of the computation.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan2( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *sqrt*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

if_freenameindex — free memory allocated by if_nameindex

SYNOPSIS

#include <net/if.h>

void if_freenameindex(struct if_nameindex **ptr*);

DESCRIPTION

The *if_freenameindex*() function shall free the memory allocated by *if_nameindex*(). The *ptr* argument shall be a pointer that was returned by *if_nameindex*(). After *if_freenameindex*() has been called, the application shall not use the array of which *ptr* is the address.

RETURN VALUE

None.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_indextoname*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( )

The Base Definitions volume of POSIX.1-2017, **<net_if.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

if_indextoname — map a network interface index to its corresponding name

SYNOPSIS

#include <net/if.h>

char *if_indextoname(unsigned *ifindex*, char **ifname*);

DESCRIPTION

The *if_indextoname*() function shall map an interface index to its corresponding name.

When this function is called, *ifname* shall point to a buffer of at least {IF_NAMESIZE} bytes. The function shall place in this buffer the name of the interface with index *ifindex*.

RETURN VALUE

If *ifindex* is an interface index, then the function shall return the value supplied in *ifname*, which points to a buffer now containing the interface name. Otherwise, the function shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *if_indextoname*() function shall fail if:

ENXIO

The interface does not exist.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_freenameindex*( ), *if_nameindex*( ), *if_nametoindex*( ), *setsockopt*( )

The Base Definitions volume of POSIX.1-2017, **<net_if.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

if_nameindex — return all network interface names and indexes

SYNOPSIS

#include <net/if.h>

struct if_nameindex **if_nameindex*(void);

DESCRIPTION

The *if_nameindex*() function shall return an array of *if_nameindex* structures, one structure per interface. The end of the array is indicated by a structure with an *if_index* field of zero and an *if_name* field of NULL.

Applications should call *if_freenameindex*() to release the memory that may be dynamically allocated by this function, after they have finished using it.

RETURN VALUE

An array of structures identifying local interfaces. A null pointer is returned upon an error, with *errno* set to indicate the error.

ERRORS

The *if_nameindex*() function may fail if:

ENOBUFS

Insufficient resources are available to complete the function.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nametoindex*( ), *setsockopt*( )

The Base Definitions volume of POSIX.1-2017, **<net_if.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

if_nametoindex — map a network interface name to its corresponding index

SYNOPSIS

#include <net/if.h>

unsigned if_nametoindex(const char **ifname*);

DESCRIPTION

The *if_nametoindex*() function shall return the interface index corresponding to name *ifname*.

RETURN VALUE

The corresponding index if *ifname* is the name of an interface; otherwise, zero.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *if_freenameindex*( ), *if_indextoname*( ), *if_nameindex*( ), *setsockopt*( )

The Base Definitions volume of POSIX.1-2017, **<net_if.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

ilogb, ilogbf, ilogbl — return an unbiased exponent

SYNOPSIS

```
#include <math.h>

int ilogb(double x);
int ilogbf(float x);
int ilogbl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall return the exponent part of their argument *x*. Formally, the return value is the integral part of $log sub{r}|x|$ as a signed integral value, for non-zero *x*, where *r* is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in *<float.h>*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent part of *x* as a signed integer value. They are equivalent to calling the corresponding *logb*() function and casting the returned value to type **int**.

If *x* is 0, the value FP_ILOGB0 shall be returned. On XSI-conformant systems, a domain error shall occur; otherwise, a domain error may occur.

If *x* is ±Inf, the value {INT_MAX} shall be returned. On XSI-conformant systems, a domain error shall occur;
otherwise, a domain error may occur.

If *x* is a NaN, the value FP_ILOGBNAN shall be returned. On XSI-conformant systems, a domain error shall occur;
otherwise, a domain error may occur.

If the correct value is greater than {INT_MAX}, a domain error shall occur and an unspecified value shall be returned. On XSI-conformant systems, a domain error shall occur and {INT_MAX} shall be returned.

If the correct value is less than {INT_MIN}, a domain error shall occur and an unspecified value shall be returned. On XSI-conformant systems, a domain error shall occur and {INT_MIN} shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The correct value is not representable as an integer.

The *x* argument is zero, NaN, or ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error

The *x* argument is zero, NaN, or ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

The errors come from taking the expected floating-point value and converting it to **int**, which is an invalid operation in IEEE Std 754-1985 (since overflow, infinity, and NaN are not representable in a type **int**), so should be a domain error.

There are no known implementations that overflow. For overflow to happen, {INT_MAX} must be less than LDBL_MAX_EXP**log2*(FLT_RADIX) or {INT_MIN} must be greater than LDBL_MIN_EXP**log2*(FLT_RADIX) if subnormals are not supported, or {INT_MIN} must be greater than (LDBL_MIN_EXP-LDBL_MANT_DIG)**log2*(FLT_RADIX) if subnormals are supported.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *logb*( ), *scalbln*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<float.h>**, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

imaxabs — return absolute value

SYNOPSIS

```
#include <inttypes.h>

intmax_t imaxabs(intmax_t j);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *imaxabs*() function shall compute the absolute value of an integer *j*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *imaxabs*() function shall return the absolute value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The absolute value of the most negative number cannot be represented in two's complement.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

imaxdiv( )

The Base Definitions volume of POSIX.1-2017, **<inttypes.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

imaxdiv — return quotient and remainder

SYNOPSIS

```
#include <inttypes.h>

imaxdiv_t imaxdiv(intmax_t numer, intmax_t denom);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *imaxdiv*() function shall compute *numer* / *denom* and *numer* % *denom* in a single operation.

RETURN VALUE

The *imaxdiv*() function shall return a structure of type **imaxdiv_t**, comprising both the quotient and the remainder. The structure shall contain (in either order) the members *quot* (the quotient) and *rem* (the remainder), each of which has type **intmax_t**.

If either part of the result cannot be represented, the behavior is undefined.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

imaxabs( )

The Base Definitions volume of POSIX.1-2017, **<inttypes.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

inet_addr, inet_ntoa — IPv4 address manipulation

SYNOPSIS

```
#include <arpa/inet.h>

in_addr_t inet_addr(const char *cp);
char *inet_ntoa(struct in_addr in);
```

DESCRIPTION

The *inet_addr*() function shall convert the string pointed to by *cp*, in the standard IPv4 dotted decimal notation, to an integer value suitable for use as an Internet address.

The *inet_ntoa*() function shall convert the Internet host address specified by *in* to a string in the Internet standard dot notation.

The *inet_ntoa*() function need not be thread-safe.

All Internet addresses shall be returned in network order (bytes ordered from left to right).

Values specified using IPv4 dotted decimal notation take one of the following forms:

a.b.c.d	When four parts are specified, each shall be interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.
a.b.c	When a three-part address is specified, the last part shall be interpreted as a 16-bit quantity and placed in the rightmost two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as **"128.net.host"**.
a.b	When a two-part address is supplied, the last part shall be interpreted as a 24-bit quantity and placed in the rightmost three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as **"net.host"**.
a	When only one part is given, the value shall be stored directly in the network address without any byte rearrangement.

All numbers supplied as parts in IPv4 dotted decimal notation may be decimal, octal, or hexadecimal, as specified in the ISO C standard (that is, a leading 0x or 0X implies hexadecimal; otherwise, a leading **'0'** implies octal; otherwise, the number is interpreted as decimal).

RETURN VALUE

Upon successful completion, *inet_addr*() shall return the Internet address. Otherwise, it shall return (**in_addr_t**)(−1).

The *inet_ntoa*() function shall return a pointer to the network address in Internet standard dot notation.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The return value of *inet_ntoa*() may point to static data that may be overwritten by subsequent calls to *inet_ntoa*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

endhostent( ), *endnetent*( )

The Base Definitions volume of POSIX.1-2017, **<arpa_inet.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

inet_ntop, inet_pton — convert IPv4 and IPv6 addresses between binary and text form

SYNOPSIS

```
#include <arpa/inet.h>

const char *inet_ntop(int af, const void *restrict src,
    char *restrict dst, socklen_t size);
int inet_pton(int af, const char *restrict src, void *restrict dst);
```

DESCRIPTION

The *inet_ntop*() function shall convert a numeric address into a text string suitable for presentation. The *af* argument shall specify the family of the address. This can be AF_INET or AF_INET6. The *src* argument points to a buffer holding an IPv4 address if the *af* argument is AF_INET, or an IPv6 address if the *af* argument is AF_INET6; the address must be in network byte order. The *dst* argument points to a buffer where the function stores the resulting text string; it shall not be NULL. The *size* argument specifies the size of this buffer, which shall be large enough to hold the text string (INET_ADDRSTRLEN characters for IPv4, INET6_ADDRSTRLEN characters for IPv6).

The *inet_pton*() function shall convert an address in its standard text presentation form into its numeric binary form. The *af* argument shall specify the family of the address. The AF_INET and AF_INET6 address families shall be supported. The *src* argument points to the string being passed in. The *dst* argument points to a buffer into which the function stores the numeric address; this shall be large enough to hold the numeric address (32 bits for AF_INET, 128 bits for AF_INET6).

If the *af* argument of *inet_pton*() is AF_INET, the *src* string shall be in the standard IPv4 dotted-decimal form:

```
ddd.ddd.ddd.ddd
```

where **"ddd"** is a one to three digit decimal number between 0 and 255 (see *inet_addr*( )). The *inet_pton*() function does not accept other formats (such as the octal numbers, hexadecimal numbers, and fewer than four numbers that *inet_addr*() accepts).

If the *af* argument of *inet_pton*() is AF_INET6, the *src* string shall be in one of the following standard IPv6 text forms:

1. The preferred form is **"x:x:x:x:x:x:x:x"**, where the **'x'**s are the hexadecimal values of the eight 16-bit pieces of the address. Leading zeros in individual fields can be omitted, but there shall be one to four hexadecimal digits in every field.
2. A string of contiguous zero fields in the preferred form can be shown as **"::"**. The **"::"** can only appear once in an address. Unspecified addresses (**"0:0:0:0:0:0:0:0"**) may be represented simply as **"::"**.
3. A third form that is sometimes more convenient when dealing with a mixed environment of IPv4 and IPv6 nodes is **"x:x:x:x:x:x:d.d.d.d"**, where the **'x'**s are the hexadecimal values of the six high-order 16-bit pieces of the address, and the **'d'**s are the decimal values of the four low-order 8-bit pieces of the address (standard IPv4 representation).

Note: A more extensive description of the standard representations of IPv6 addresses can be found in RFC 2373.

RETURN VALUE

The *inet_ntop*() function shall return a pointer to the buffer containing the text string if the conversion succeeds, and NULL otherwise, and set *errno* to indicate the error.

The *inet_pton*() function shall return 1 if the conversion succeeds, with the address pointed to by *dst* in network byte order. It shall return 0 if the input is not a valid IPv4 dotted-decimal string or a valid IPv6 address string, or −1 with *errno* set to **[EAFNOSUPPORT]** if the *af* argument is unknown.

ERRORS

The *inet_ntop*() and *inet_pton*() functions shall fail if:

EAFNOSUPPORT
The *af* argument is invalid.

ENOSPC
The size of the *inet_ntop*() result buffer is inadequate.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<arpa_inet.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

initstate, random, setstate, srandom — pseudo-random number functions

SYNOPSIS

#include <stdlib.h>

char *initstate(unsigned *seed*, char **state*, size_t *size*);
long random(void);
char *setstate(char **state*);
void srandom(unsigned *seed*);

DESCRIPTION

The *random*() function shall use a non-linear additive feedback random-number generator employing a default state array size of 31 **long** integers to return successive pseudo-random numbers in the range from 0 to $2^{31}-1$. The period of this random-number generator is approximately 16 x ($2^{31}-1$). The size of the state array determines the period of the random-number generator. Increasing the state array size shall increase the period.

With 256 bytes of state information, the period of the random-number generator shall be greater than 2^{69}.

Like *rand*(), *random*() shall produce by default a sequence of numbers that can be duplicated by calling *srandom*() with 1 as the seed.

The *srandom*() function shall initialize the current state array using the value of *seed*.

The *initstate*() and *setstate*() functions handle restarting and changing random-number generators. The *initstate*() function allows a state array, pointed to by the *state* argument, to be initialized for future use. The *size* argument, which specifies the size in bytes of the state array, shall be used by *initstate*() to decide what type of random-number generator to use; the larger the state array, the more random the numbers. Values for the amount of state information are 8, 32, 64, 128, and 256 bytes. Other values greater than 8 bytes are rounded down to the nearest one of these values. If *initstate*() is called with $8 \leq size < 32$, then *random*() shall use a simple linear congruential random number generator. The *seed* argument specifies a starting point for the random-number sequence and provides for restarting at the same point. The *initstate*() function shall return a pointer to the previous state information array.

If *initstate*() has not been called, then *random*() shall behave as though *initstate*() had been called with *seed*=1 and *size*=128.

Once a state has been initialized, *setstate*() allows switching between state arrays. The array defined by the *state* argument shall be used for further random-number generation until *initstate*() is called or *setstate*() is called again. The *setstate*() function shall return a pointer to the previous state array.

RETURN VALUE

If *initstate*() is called with *size* less than 8, it shall return NULL.

The *random*() function shall return the generated pseudo-random number.

The *srandom*() function shall not return a value.

Upon successful completion, *initstate*() and *setstate*() shall return a pointer to the previous state array; otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After initialization, a state array can be restarted at a different point in one of two ways:

1. The *initstate*() function can be used, with the desired seed, state array, and size of the array.
2. The *setstate*() function, with the desired state, can be used, followed by *srandom*() with the desired seed. The advantage of using both of these functions is that the size of the state array does not have to be saved once it is initialized.

Although some implementations of *random*() have written messages to standard error, such implementations do not conform to POSIX.1-2008.

Issue 5 restored the historical behavior of this function.

Threaded applications should use *erand48*(), *nrand48*(), or *jrand48*() instead of *random*() when an independent random number sequence in multiple threads is required.

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

drand48( ), *rand*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

insque, remque — insert or remove an element in a queue

SYNOPSIS

```
#include <search.h>

void insque(void *element, void *pred);
void remque(void *element);
```

DESCRIPTION

The *insque*() and *remque*() functions shall manipulate queues built from doubly-linked lists. The queue can be either circular or linear. An application using *insque*() or *remque*() shall ensure it defines a structure in which the first two members of the structure are pointers to the same type of structure, and any further members are application-specific. The first member of the structure is a forward pointer to the next entry in the queue. The second member is a backward pointer to the previous entry in the queue. If the queue is linear, the queue is terminated with null pointers. The names of the structure and of the pointer members are not subject to any special restriction.

The *insque*() function shall insert the element pointed to by *element* into a queue immediately after the element pointed to by *pred*.

The *remque*() function shall remove the element pointed to by *element* from a queue.

If the queue is to be used as a linear list, invoking *insque*(&*element*, NULL), where *element* is the initial element of the queue, shall initialize the forward and backward pointers of *element* to null pointers.

If the queue is to be used as a circular list, the application shall ensure it initializes the forward pointer and the backward pointer of the initial element of the queue to the element's own address.

RETURN VALUE

The *insque*() and *remque*() functions do not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Creating a Linear Linked List

The following example creates a linear linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

insque (&element1, NULL);
insque (&element2, &element1);
```

Creating a Circular Linked List

The following example creates a circular linked list.

```
#include <search.h>
...
struct myque element1;
struct myque element2;

char *data1 = "DATA1";
char *data2 = "DATA2";
...
element1.data = data1;
element2.data = data2;

element1.fwd = &element1;
element1.bck = &element1;

insque (&element2, &element1);
```

Removing an Element

The following example removes the element pointed to by *element1*.

```
#include <search.h>
...
struct myque element1;
...
remque (&element1);
```

APPLICATION USAGE

The historical implementations of these functions described the arguments as being of type **struct qelem *** rather than as being of type **void *** as defined here. In those implementations, **struct qelem** was commonly defined in *<search.h>* as:

```
struct qelem {
    struct qelem  *q_forw;
    struct qelem  *q_back;
};
```

Applications using these functions, however, were never able to use this structure directly since it provided no room for the actual data contained in the elements. Most applications defined structures that contained the two pointers as the initial elements and also provided space for, or pointers to, the object's data. Applications that used these functions to update more than one type of table also had the problem of specifying two or more different structures with the same name, if they literally used **struct qelem** as specified.

As described here, the implementations were actually expecting a structure type where the first two members were forward and backward pointers to structures. With C compilers that didn't provide function prototypes, applications used structures as specified in the DESCRIPTION above and the compiler did what the application expected.

If this method had been carried forward with an ISO C standard compiler and the historical function prototype, most applications would have to be modified to cast pointers to the structures actually used to be pointers to **struct qelem** to avoid compilation warnings. By specifying **void *** as the argument type, applications do not need to change (unless they specifically referenced **struct qelem** and depended on it being defined in *<search.h>*).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<search.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ioctl — control a STREAMS device (**STREAMS**)

SYNOPSIS

#include <stropts.h>

int ioctl(int *fildes*, int *request*, ... /* arg */);

DESCRIPTION

The *ioctl*() function shall perform a variety of control functions on STREAMS devices. For non-STREAMS devices, the functions performed by this call are unspecified. The *request* argument and an optional third argument (with varying type) shall be passed to and interpreted by the appropriate part of the STREAM associated with *fildes*.

The *fildes* argument is an open file descriptor that refers to a device.

The *request* argument selects the control function to be performed and shall depend on the STREAMS device being addressed.

The *arg* argument represents additional information that is needed by this specific STREAMS device to perform the requested function. The type of *arg* depends upon the particular control request, but it shall be either an integer or a pointer to a device-specific data structure.

The *ioctl*() commands applicable to STREAMS, their arguments, and error conditions that apply to each individual command are described below.

The following *ioctl*() commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH
: Pushes the module whose name is pointed to by *arg* onto the top of the current STREAM, just below the STREAM head. It then calls the *open*() function of the newly-pushed module.

  The *ioctl*() function with the I_PUSH command shall fail if:

  EINVAL
  : Invalid module name.

  ENXIO
  : Open function of new module failed.

  ENXIO
  : Hangup received on *fildes*.

I_POP
: Removes the module just below the STREAM head of the STREAM pointed to by *fildes*. The *arg* argument should be 0 in an I_POP request.

  The *ioctl*() function with the I_POP command shall fail if:

  EINVAL
  : No module present in the STREAM.

  ENXIO
  : Hangup received on *fildes*.

I_LOOK
: Retrieves the name of the module just below the STREAM head of the STREAM pointed to by *fildes*, and places it in a character string pointed to by *arg*. The buffer pointed to by *arg* should be at least FMNAMESZ+1 bytes long, where FMNAMESZ is defined in *<stropts.h>*.

  The *ioctl*() function with the I_LOOK command shall fail if:

EINVAL
No module present in the STREAM.

I_FLUSH Flushes read and/or write queues, depending on the value of *arg*. Valid *arg* values are:

FLUSHR	Flush all read queues.
FLUSHW	Flush all write queues.
FLUSHRW	Flush all read and all write queues.

The *ioctl*() function with the I_FLUSH command shall fail if:

EINVAL
Invalid *arg* value.

EAGAIN or **ENOSR**
Unable to allocate buffers for flush message.

ENXIO
Hangup received on *fildes*.

I_FLUSHBAND
Flushes a particular band of messages. The *arg* argument points to a **bandinfo** structure. The *bi_flag* member may be one of FLUSHR, FLUSHW, or FLUSHRW as described above. The *bi_pri* member determines the priority band to be flushed.

I_SETSIG Requests that the STREAMS implementation send the SIGPOLL signal to the calling process when a particular event has occurred on the STREAM associated with *fildes*. I_SETSIG supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the process should be signaled. It is the bitwise-inclusive OR of any combination of the following constants:

S_RDNORM	A normal (priority band set to 0) message has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
S_RDBAND	A message with a non-zero priority band has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
S_INPUT	A message, other than a high-priority message, has arrived at the head of a STREAM head read queue. A signal shall be generated even if the message is of zero length.
S_HIPRI	A high-priority message is present on a STREAM head read queue. A signal shall be generated even if the message is of zero length.
S_OUTPUT	The write queue for normal data (priority band 0) just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) normal data downstream.
S_WRNORM	Equivalent to S_OUTPUT.
S_WRBAND	The write queue for a non-zero priority band just below the STREAM head is no longer full. This notifies the process that there is room on the queue for sending (or writing) priority data downstream.
S_MSG	A STREAMS signal message that contains the SIGPOLL signal has reached the front of the STREAM head read queue.
S_ERROR	Notification of an error condition has reached the STREAM head.
S_HANGUP	Notification of a hangup has reached the STREAM head.

S_BANDURG
: When used in conjunction with S_RDBAND, SIGURG is generated instead of SIGPOLL when a priority message reaches the front of the STREAM head read queue.

If *arg* is 0, the calling process shall be unregistered and shall not receive further SIGPOLL signals for the stream associated with *fildes*.

Processes that wish to receive SIGPOLL signals shall ensure that they explicitly register to receive them using I_SETSIG. If several processes register to receive this signal for the same event on the same STREAM, each process shall be signaled when the event occurs.

The *ioctl*() function with the I_SETSIG command shall fail if:

EINVAL
The value of *arg* is invalid.

EINVAL
The value of *arg* is 0 and the calling process is not registered to receive the SIGPOLL signal.

EAGAIN
There were insufficient resources to store the signal request.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a SIGPOLL signal. The events are returned as a bitmask in an **int** pointed to by *arg*, where the events are those specified in the description of I_SETSIG above.

The *ioctl*() function with the I_GETSIG command shall fail if:

EINVAL
Process is not registered to receive the SIGPOLL signal.

I_FIND Compares the names of all modules currently present in the STREAM to the name pointed to by *arg*, and returns 1 if the named module is present in the STREAM, or returns 0 if the named module is not present.

The *ioctl*() function with the I_FIND command shall fail if:

EINVAL
arg does not contain a valid module name.

I_PEEK Retrieves the information in the first message on the STREAM head read queue without taking the message off the queue. It is analogous to *getmsg*() except that this command does not remove the message from the queue. The *arg* argument points to a **strpeek** structure.

The application shall ensure that the *maxlen* member in the **ctlbuf** and **databuf strbuf** structures is set to the number of bytes of control information and/or data information, respectively, to retrieve. The *flags* member may be marked RS_HIPRI or 0, as described by *getmsg*(). If the process sets *flags* to RS_HIPRI, for example, I_PEEK shall only look for a high-priority message on the STREAM head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the STREAM head read queue, or if the RS_HIPRI flag was set in *flags* and a high-priority message was not present on the STREAM head read queue. It does not wait for a message to arrive. On return, **ctlbuf** specifies information in the control buffer, **databuf** specifies information in the data buffer, and *flags* contains the value RS_HIPRI or 0.

I_SRDOPT Sets the read mode using the value of the argument *arg*. Read modes are described in *read*(). Valid *arg* flags are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.

RMSGN Message-nondiscard mode.

The bitwise-inclusive OR of RMSGD and RMSGN shall return **[EINVAL]**. The bitwise-inclusive OR of RNORM and either RMSGD or RMSGN shall result in the other flag overriding RNORM which is the default.

In addition, treatment of control messages by the STREAM head may be changed by setting any of the following flags in *arg*:

RPROTNORM
: Fail *read*() with **[EBADMSG]** if a message containing a control part is at the front of the STREAM head read queue.

RPROTDAT Deliver the control part of a message as data when a process issues a *read*().

RPROTDIS Discard the control part of a message, delivering any data portion, when a process issues a *read*().

The *ioctl*() function with the I_SRDOPT command shall fail if:

EINVAL
The *arg* argument is not valid.

I_GRDOPT Returns the current read mode setting, as described above, in an **int** pointed to by the argument *arg*. Read modes are described in *read*().

I_NREAD Counts the number of data bytes in the data part of the first message on the STREAM head read queue and places this value in the **int** pointed to by *arg*. The return value for the command shall be the number of messages on the STREAM head read queue. For example, if 0 is returned in *arg*, but the *ioctl*() return value is greater than 0, this indicates that a zero-length message is next on the queue.

I_FDINSERT
Creates a message from specified buffer(s), adds information about another STREAM, and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below. The *arg* argument points to a **strfdinsert** structure.

The application shall ensure that the *len* member in the **ctlbuf strbuf** structure is set to the size of a **t_uscalar_t** plus the number of bytes of control information to be sent with the message. The *fildes* member specifies the file descriptor of the other STREAM, and the *offset* member, which must be suitably aligned for use as a **t_uscalar_t**, specifies the offset from the start of the control buffer where I_FDINSERT shall store a **t_uscalar_t** whose interpretation is specific to the STREAM end. The application shall ensure that the *len* member in the **databuf strbuf** structure is set to the number of bytes of data information to be sent with the message, or to 0 if no data part is to be sent.

The *flags* member specifies the type of message to be created. A normal message is created if *flags* is set to 0, and a high-priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT shall block if the STREAM write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NONBLOCK is set. Instead, it fails and sets *errno* to **[EAGAIN]**.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message is sent.

The *ioctl*() function with the I_FDINSERT command shall fail if:

EAGAIN
A non-priority message is specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions.

EAGAIN or **ENOSR**
Buffers cannot be allocated for the message that is to be created.

EINVAL
One of the following:

-- The *fildes* member of the **strfdinsert** structure is not a valid, open STREAM file descriptor.

-- The size of a **t_uscalar_t** plus *offset* is greater than the *len* member for the buffer specified through **ctlbuf**.

-- The *offset* member does not specify a properly-aligned location in the data buffer.

-- An undefined value is stored in *flags*.

ENXIO
Hangup received on the STREAM identified by either the *fildes* argument or the *fildes* member of the **strfdinsert** structure.

ERANGE
The *len* member for the buffer specified through **databuf** does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module; or the *len* member for the buffer specified through **databuf** is larger than the maximum configured size of the data part of a message; or the *len* member for the buffer specified through **ctlbuf** is larger than the maximum configured size of the control part of a message.

I_STR Constructs an internal STREAMS *ioctl*() message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send *ioctl*() requests to downstream modules and drivers. It allows information to be sent with *ioctl*(), and returns to the process any information sent upstream by the downstream recipient. I_STR shall block until the system responds with either a positive or negative acknowledgement message, or until the request times out after some period of time. If the request times out, it shall fail with *errno* set to **[ETIME]**.

At most, one I_STR can be active on a STREAM. Further I_STR calls shall block until the active I_STR completes at the STREAM head. The default timeout interval for these requests is 15 seconds. The O_NONBLOCK flag has no effect on this call.

To send requests downstream, the application shall ensure that *arg* points to a **strioctl** structure.

The *ic_cmd* member is the internal *ioctl*() command intended for a downstream module or driver and *ic_timout* is the number of seconds (−1=infinite, 0=use implementation-defined timeout interval, >0=as specified) an I_STR request shall wait for acknowledgement before timing out. *ic_len* is the number of bytes in the data argument, and *ic_dp* is a pointer to the data argument. The *ic_len* member has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the process (the buffer pointed to by *ic_dp* should be large enough to contain the maximum amount of data that any module or the driver in the STREAM can return).

The STREAM head shall convert the information pointed to by the **strioctl** structure to an internal *ioctl*() command message and send it downstream.

The *ioctl*() function with the I_STR command shall fail if:

EAGAIN or **ENOSR**
Unable to allocate buffers for the *ioctl*() message.

EINVAL
The *ic_len* member is less than 0 or larger than the maximum configured size of the data part of a message, or *ic_timout* is less than −1.

ENXIO
Hangup received on *fildes*.

ETIME
A downstream *ioctl*() timed out before acknowledgement was received.

An I_STR can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the STREAM head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the *ioctl*() command sent downstream fails. For these cases, I_STR shall fail with *errno* set to the value in the message.

I_SWROPT Sets the write mode using the value of the argument *arg*. Valid bit settings for *arg* are:

SNDZERO Send a zero-length message downstream when a *write*() of 0 bytes occurs. To not send a zero-length message when a *write*() of 0 bytes occurs, the application shall ensure that this bit is not set in *arg* (for example, *arg* would be set to 0).

The *ioctl*() function with the I_SWROPT command shall fail if:

EINVAL
arg is not the above value.

I_GWROPT Returns the current write mode setting, as described above, in the **int** that is pointed to by the argument *arg*.

I_SENDFD Creates a new reference to the open file description associated with the file descriptor *arg*, and writes a message on the STREAMS-based pipe *fildes* containing this reference, together with the user ID and group ID of the calling process.

The *ioctl*() function with the I_SENDFD command shall fail if:

EAGAIN
The sending STREAM is unable to allocate a message block to contain the file pointer; or the read queue of the receiving STREAM head is full and cannot accept the message sent by I_SENDFD.

EBADF
The *arg* argument is not a valid, open file descriptor.

EINVAL
The *fildes* argument is not connected to a STREAM pipe.

ENXIO
Hangup received on *fildes*.

The *ioctl*() function with the I_SENDFD command may fail if:

EINVAL
The *arg* argument is equal to the *fildes* argument.

I_RECVFD Retrieves the reference to an open file description from a message written to a STREAMS-based pipe using the I_SENDFD command, and allocates a new file descriptor in the calling process that refers to this open file description. The *arg* argument is a pointer to a **strrecvfd** data structure as defined in <*stropts.h*>.

The *fd* member is a file descriptor. The *uid* and *gid* members are the effective user ID and effective group ID, respectively, of the sending process.

If O_NONBLOCK is not set, I_RECVFD shall block until a message is present at the STREAM head. If O_NONBLOCK is set, I_RECVFD shall fail with *errno* set to

[EAGAIN] if no message is present at the STREAM head.

If the message at the STREAM head is a message sent by an I_SENDFD, a new file descriptor shall be allocated for the open file descriptor referenced in the message. The new file descriptor is placed in the *fd* member of the **strrecvfd** structure pointed to by *arg*.

The *ioctl*() function with the I_RECVFD command shall fail if:

EAGAIN
A message is not present at the STREAM head read queue and the O_NONBLOCK flag is set.

EBADMSG
The message at the STREAM head read queue is not a message containing a passed file descriptor.

EMFILE
All file descriptors available to the process are currently open.

ENXIO
Hangup received on *fildes*.

I_LIST Allows the process to list all the module names on the STREAM, up to and including the topmost driver name. If *arg* is a null pointer, the return value shall be the number of modules, including the driver, that are on the STREAM pointed to by *fildes*. This lets the process allocate enough space for the module names. Otherwise, it should point to a **str_list** structure.

The *sl_nmods* member indicates the number of entries the process has allocated in the array. Upon return, the *sl_modlist* member of the **str_list** structure shall contain the list of module names, and the number of entries that have been filled into the *sl_modlist* array is found in the *sl_nmods* member (the number includes the number of modules including the driver). The return value from *ioctl*() shall be 0. The entries are filled in starting at the top of the STREAM and continuing downstream until either the end of the STREAM is reached, or the number of requested modules (*sl_nmods*) is satisfied.

The *ioctl*() function with the I_LIST command shall fail if:

EINVAL
The *sl_nmods* member is less than 1.

EAGAIN or **ENOSR**
Unable to allocate buffers.

I_ATMARK Allows the process to see if the message at the head of the STREAM head read queue is marked by some module downstream. The *arg* argument determines how the checking is done when there may be multiple marked messages on the STREAM head read queue. It may take on the following values:

ANYMARK Check if the message is marked.

LASTMARK Check if the message is the last one marked on the queue.

The bitwise-inclusive OR of the flags ANYMARK and LASTMARK is permitted.

The return value shall be 1 if the mark condition is satisfied; otherwise, the value shall be 0.

The *ioctl*() function with the I_ATMARK command shall fail if:

EINVAL
Invalid *arg* value.

I_CKBAND Checks if the message of a given priority band exists on the STREAM head read queue. This shall return 1 if a message of the given priority exists, 0 if no such message exists, or −1 on error. *arg* should be of type **int**.

The *ioctl*() function with the I_CKBAND command shall fail if:

EINVAL
Invalid *arg* value.

I_GETBAND
Returns the priority band of the first message on the STREAM head read queue in the integer referenced by *arg*.

The *ioctl*() function with the I_GETBAND command shall fail if:

ENODATA
No message on the STREAM head read queue.

I_CANPUT Checks if a certain band is writable. *arg* is set to the priority band in question. The return value shall be 0 if the band is flow-controlled, 1 if the band is writable, or −1 on error.

The *ioctl*() function with the I_CANPUT command shall fail if:

EINVAL
Invalid *arg* value.

I_SETCLTIME
This request allows the process to set the time the STREAM head shall delay when a STREAM is closing and there is data on the write queues. Before closing each module or driver, if there is data on its write queue, the STREAM head shall delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, it shall be flushed. The *arg* argument is a pointer to an integer specifying the number of milliseconds to delay, rounded up to the nearest valid value. If I_SETCLTIME is not performed on a STREAM, an implementation-defined default timeout interval is used.

The *ioctl*() function with the I_SETCLTIME command shall fail if:

EINVAL
Invalid *arg* value.

I_GETCLTIME
Returns the close time delay in the integer pointed to by *arg*.

Multiplexed STREAMS Configurations

The following commands are used for connecting and disconnecting multiplexed STREAMS configurations. These commands use an implementation-defined default timeout interval.

I_LINK Connects two STREAMs, where *fildes* is the file descriptor of the STREAM connected to the multiplexing driver, and *arg* is the file descriptor of the STREAM connected to another driver. The STREAM designated by *arg* is connected below the multiplexing driver. I_LINK requires the multiplexing driver to send an acknowledgement message to the STREAM head regarding the connection. This call shall return a multiplexer ID number (an identifier used to disconnect the multiplexer; see I_UNLINK) on success, and −1 on failure.

The *ioctl*() function with the I_LINK command shall fail if:

ENXIO
Hangup received on *fildes*.

ETIME
Timeout before acknowledgement message was received at STREAM head.

EAGAIN or **ENOSR**
Unable to allocate STREAMS storage to perform the I_LINK.

EBADF
The *arg* argument is not a valid, open file descriptor.

EINVAL
The *fildes* argument does not support multiplexing; or *arg* is not a STREAM or is already connected downstream from a multiplexer; or the specified I_LINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.

An I_LINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_LINK fails with *errno* set to the value in the message.

I_UNLINK Disconnects the two STREAMs specified by *fildes* and *arg*. *fildes* is the file descriptor of the STREAM connected to the multiplexing driver. The *arg* argument is the multiplexer ID number that was returned by the I_LINK *ioctl*() command when a STREAM was connected downstream from the multiplexing driver. If *arg* is MUXID_ALL, then all STREAMs that were connected to *fildes* shall be disconnected. As in I_LINK, this command requires acknowledgement.

The *ioctl*() function with the I_UNLINK command shall fail if:

ENXIO
Hangup received on *fildes*.

ETIME
Timeout before acknowledgement message was received at STREAM head.

EAGAIN or **ENOSR**
Unable to allocate buffers for the acknowledgement message.

EINVAL
Invalid multiplexer ID number.

An I_UNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_UNLINK shall fail with *errno* set to the value in the message.

I_PLINK Creates a *persistent connection* between two STREAMs, where *fildes* is the file descriptor of the STREAM connected to the multiplexing driver, and *arg* is the file descriptor of the STREAM connected to another driver. This call shall create a persistent connection which can exist even if the file descriptor *fildes* associated with the upper STREAM to the multiplexing driver is closed. The STREAM designated by *arg* gets connected via a persistent connection below the multiplexing driver. I_PLINK requires the multiplexing driver to send an acknowledgement message to the STREAM head. This call shall return a multiplexer ID number (an identifier that may be used to disconnect the multiplexer; see I_PUNLINK) on success, and −1 on failure.

The *ioctl*() function with the I_PLINK command shall fail if:

ENXIO
Hangup received on *fildes*.

ETIME
Timeout before acknowledgement message was received at STREAM head.

EAGAIN or **ENOSR**
Unable to allocate STREAMS storage to perform the I_PLINK.

EBADF
The *arg* argument is not a valid, open file descriptor.

EINVAL
The *fildes* argument does not support multiplexing; or *arg* is not a STREAM or is already connected downstream from a multiplexer; or the specified I_PLINK operation would connect the STREAM head in more than one place in the multiplexed STREAM.

An I_PLINK can also fail while waiting for the multiplexing driver to acknowledge the request, if a message indicating an error or a hangup is received at the STREAM head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PLINK shall fail with *errno* set to the value in the message.

I_PUNLINK Disconnects the two STREAMs specified by *fildes* and *arg* from a persistent connection. The *fildes* argument is the file descriptor of the STREAM connected to the multiplexing driver. The *arg* argument is the multiplexer ID number that was returned by the I_PLINK *ioctl*() command when a STREAM was connected downstream from the multiplexing driver. If *arg* is MUXID_ALL, then all STREAMs which are persistent connections to *fildes* shall be disconnected. As in I_PLINK, this command requires the multiplexing driver to acknowledge the request.

The *ioctl*() function with the I_PUNLINK command shall fail if:

ENXIO
Hangup received on *fildes*.

ETIME
Timeout before acknowledgement message was received at STREAM head.

EAGAIN or **ENOSR**
Unable to allocate buffers for the acknowledgement message.

EINVAL
Invalid multiplexer ID number.

An I_PUNLINK can also fail while waiting for the multiplexing driver to acknowledge the request if a message indicating an error or a hangup is received at the STREAM head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, I_PUNLINK shall fail with *errno* set to the value in the message.

RETURN VALUE

Upon successful completion, *ioctl*() shall return a value other than −1 that depends upon the STREAMS device control function. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

Under the following general conditions, *ioctl*() shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor.

EINTR
A signal was caught during the *ioctl*() operation.

EINVAL
The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

If an underlying device driver detects an error, then *ioctl*() shall fail if:

EINVAL
The *request* or *arg* argument is not valid for this device.

EIO Some physical I/O error has occurred.

ENOTTY
The file associated with the *fildes* argument is not a STREAMS device that accepts control functions.

ENXIO
The *request* and *arg* arguments are valid for this device driver, but the service requested cannot be performed on this particular sub-device.

ENODEV
The *fildes* argument refers to a valid STREAMS device, but the corresponding device driver does not support the *ioctl*() function.

If a STREAM is connected downstream from a multiplexer, any *ioctl*() command except I_UNLINK and I_PUNLINK shall set *errno* to **[EINVAL]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The implementation-defined timeout interval for STREAMS has historically been 15 seconds.

RATIONALE

None.

FUTURE DIRECTIONS

The *ioctl*() function may be removed in a future version.

SEE ALSO

Section 2.6, *STREAMS*, *close*( ), *fcntl*( ), *getmsg*( ), *open*( ), *pipe*( ), *poll*( ), *putmsg*( ), *read*( ), *sigaction*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isalnum, isalnum_l — test for an alphanumeric character

SYNOPSIS

#include <ctype.h>

int isalnum(int *c*);
int isalnum_l(int *c*, locale_t *locale*);

DESCRIPTION

For *isalnum*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isalnum*() and *isalnum_l*() functions shall test whether *c* is a character of class **alpha** or **digit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isalnum_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isalnum*() and *isalnum_l*() functions shall return non-zero if *c* is an alphanumeric character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalpha( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isalpha, isalpha_l — test for an alphabetic character

SYNOPSIS

```
#include <ctype.h>

int isalpha(int c);
int isalpha_l(int c, locale_t locale);
```

DESCRIPTION

For *isalpha*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isalpha*() and *isalpha_l*() functions shall test whether *c* is a character of class **alpha** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isalpha_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isalpha*() and *isalpha_l*() functions shall return non-zero if *c* is an alphabetic character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isascii — test for a 7-bit US-ASCII character

SYNOPSIS

#include <ctype.h>

int isascii(int *c*);

DESCRIPTION

The *isascii*() function shall test whether *c* is a 7-bit US-ASCII character code.

The *isascii*() function is defined on all integer values.

RETURN VALUE

The *isascii*() function shall return non-zero if *c* is a 7-bit US-ASCII character code between 0 and octal 0177 inclusive; otherwise, it shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *isascii*() function cannot be used portably in a localized application.

RATIONALE

None.

FUTURE DIRECTIONS

The *isascii*() function may be removed in a future version.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<ctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isastream — test a file descriptor (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int isastream(int fildes);
```

DESCRIPTION

The *isastream*() function shall test whether *fildes*, an open file descriptor, is associated with a STREAMS-based file.

RETURN VALUE

Upon successful completion, *isastream*() shall return 1 if *fildes* refers to a STREAMS-based file and 0 if not. Otherwise, *isastream*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *isastream*() function shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *isastream*() function may be removed in a future version.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isatty — test for a terminal device

SYNOPSIS

#include <unistd.h>

int isatty(int *fildes*);

DESCRIPTION

The *isatty*() function shall test whether *fildes*, an open file descriptor, is associated with a terminal device.

RETURN VALUE

The *isatty*() function shall return 1 if *fildes* is associated with a terminal; otherwise, it shall return 0 and may set *errno* to indicate the error.

ERRORS

The *isatty*() function may fail if:

EBADF
The *fildes* argument is not a valid open file descriptor.

ENOTTY
The file associated with the *fildes* argument is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *isatty*() function does not necessarily indicate that a human being is available for interaction via *fildes*. It is quite possible that non-terminal devices are connected to the communications line.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isblank, isblank_l — test for a blank character

SYNOPSIS

```
#include <ctype.h>

int isblank(int c);
int isblank_l(int c, locale_t locale);
```

DESCRIPTION

For *isblank*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isblank*() and *isblank_l*() functions shall test whether *c* is a character of class **blank** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isblank_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isblank*() and *isblank_l*() functions shall return non-zero if *c* is a <blank>; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iscntrl, iscntrl_l — test for a control character

SYNOPSIS

```
#include <ctype.h>

int iscntrl(int c);
int iscntrl_l(int c, locale_t locale);
```

DESCRIPTION

For *iscntrl*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iscntrl*() and *iscntrl_l*() functions shall test whether *c* is a character of class **cntrl** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is a type **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iscntrl_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iscntrl*() and *iscntrl_l*() functions shall return non-zero if *c* is a control character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isdigit, isdigit_l — test for a decimal digit

SYNOPSIS

```
#include <ctype.h>

int isdigit(int c);
int isdigit_l(int c, locale_t locale);
```

DESCRIPTION

For *isdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isdigit*() and *isdigit_l*() functions shall test whether *c* is a character of class **digit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isdigit*() and *isdigit_l*() functions shall return non-zero if *c* is a decimal digit; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isfinite — test for finite value

SYNOPSIS

```
#include <math.h>

int isfinite(real-floating x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isfinite*() macro shall determine whether its argument has a finite value (zero, subnormal, or normal, and not infinite or NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isfinite*() macro shall return a non-zero value if and only if its argument has a finite value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isinf*( ), *isnan*( ), *isnormal*( ), *signbit*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isgraph, isgraph_l — test for a visible character

SYNOPSIS

#include <ctype.h>

int isgraph(int *c*);
int isgraph_l(int *c*, locale_t *locale*);

DESCRIPTION

For *isgraph*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isgraph*() and *isgraph_l*() functions shall test whether *c* is a character of class **graph** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isgraph_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isgraph*() and *isgraph_l*() functions shall return non-zero if *c* is a character with a visible representation; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isgreater — test if x greater than y

SYNOPSIS

#include <math.h>

int isgreater(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isgreater*() macro shall determine whether its first argument is greater than its second argument. The value of *isgreater*(*x*, *y*) shall be equal to (*x*) > (*y*); however, unlike (*x*) > (*y*), *isgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *isgreater*() macro shall return the value of (*x*) > (*y*).

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreaterequal( ), *isless*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isgreaterequal — test if x is greater than or equal to y

SYNOPSIS

#include <math.h>

int isgreaterequal(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isgreaterequal*() macro shall determine whether its first argument is greater than or equal to its second argument. The value of *isgreaterequal*(*x*, *y*) shall be equal to (x) ≥ (y); however, unlike (x) ≥ (y), *isgreaterequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *isgreaterequal*() macro shall return the value of (x) ≥ (y).

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( ), *isless*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isinf — test for infinity

SYNOPSIS

```
#include <math.h>

int isinf(real-floating x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isinf*() macro shall determine whether its argument value is an infinity (positive or negative). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isinf*() macro shall return a non-zero value if and only if its argument has an infinite value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isnan*( ), *isnormal*( ), *signbit*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isless — test if x is less than y

SYNOPSIS

#include <math.h>

int isless(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isless*() macro shall determine whether its first argument is less than its second argument. The value of *isless*(*x*, *y*) shall be equal to (*x*) < (*y*); however, unlike (*x*) < (*y*), *isless*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *isless*() macro shall return the value of (*x*) < (*y*).

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( ), *isgreaterequal*( ), *islessequal*( ), *islessgreater*( ), *isunordered*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

islessequal — test if x is less than or equal to y

SYNOPSIS

#include <math.h>

int islessequal(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *islessequal*() macro shall determine whether its first argument is less than or equal to its second argument. The value of *islessequal*(*x*, *y*) shall be equal to (*x*) <= (*y*); however, unlike (*x*) <= (*y*), *islessequal*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered.

RETURN VALUE

Upon successful completion, the *islessequal*() macro shall return the value of (*x*) <= (*y*).

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( ), *isgreaterequal*( ), *isless*( ), *islessgreater*( ), *isunordered*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

islessgreater — test if x is less than or greater than y

SYNOPSIS

#include <math.h>

int islessgreater(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *islessgreater*() macro shall determine whether its first argument is less than or greater than its second argument. The *islessgreater*(*x*, *y*) macro is similar to (*x*) < (*y*) || (*x*) > (*y*); however, *islessgreater*(*x*, *y*) shall not raise the invalid floating-point exception when *x* and *y* are unordered (nor shall it evaluate *x* and *y* twice).

RETURN VALUE

Upon successful completion, the *islessgreater*() macro shall return the value of (*x*) < (*y*) || (*x*) > (*y*).

If *x* or *y* is NaN, 0 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( ), *isgreaterequal*( ), *isless*( ), *islessequal*( ), *isunordered*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

islower, islower_l — test for a lowercase letter

SYNOPSIS

#include <ctype.h>

int islower(int *c*);
int islower_l(int *c*, locale_t *locale*);

DESCRIPTION

For *islower*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *islower*() and *islower_l*() functions shall test whether *c* is a character of class **lower** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *islower_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *islower*() and *islower_l*() functions shall return non-zero if *c* is a lowercase letter; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Testing for a Lowercase Letter

Two examples follow, the first using *islower*(), the second using multiple concurrent locales and *islower_l*().

The examples test whether the value is a lowercase letter, based on the current locale, then use it as part of a key value.

```
/* Example 1 -- using islower() */
#include <ctype.h>
#include <stdlib.h>
#include <locale.h>
...
char *keystr;
int elementlen, len;
unsigned char c;
...
setlocale(LC_ALL, "");
...
len = 0;
while (len < elementlen) {
    c = (unsigned char) (rand() % 256);
...
   if (islower(c))
      keystr[len++] = c;
   }
...

/* Example 2 -- using islower_l() */
#include <ctype.h>
#include <stdlib.h>
#include <locale.h>
...
char *keystr;
int elementlen, len;
unsigned char c;
...
locale_t loc = newlocale (LC_ALL_MASK, "", (locale_t) 0);
...
len = 0;
while (len < elementlen) {
   c = (unsigned char) (rand() % 256);
...
   if (islower_l(c, loc))
      keystr[len++] = c;
   }
...
```

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isnan — test for a NaN

SYNOPSIS

```
#include <math.h>

int isnan(real-floating x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isnan*() macro shall determine whether its argument value is a NaN. First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isnan*() macro shall return a non-zero value if and only if its argument has a NaN value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnormal*( ), *signbit*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isnormal — test for a normal value

SYNOPSIS

```
#include <math.h>

int isnormal(real-floating x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isnormal*() macro shall determine whether its argument value is normal (neither zero, subnormal, infinite, nor NaN). First, an argument represented in a format wider than its semantic type is converted to its semantic type. Then determination is based on the type of the argument.

RETURN VALUE

The *isnormal*() macro shall return a non-zero value if and only if its argument has a normal value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnan*( ), *signbit*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isprint, isprint_l — test for a printable character

SYNOPSIS

```
#include <ctype.h>

int isprint(int c);
int isprint_l(int c, locale_t locale);
```

DESCRIPTION

For *isprint*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isprint*() and *isprint_l*() functions shall test whether *c* is a character of class **print** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isprint_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isprint*() and *isprint_l*() functions shall return non-zero if *c* is a printable character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ispunct, ispunct_l — test for a punctuation character

SYNOPSIS

```
#include <ctype.h>

int ispunct(int c);
int ispunct_l(int c, locale_t locale);
```

DESCRIPTION

For *ispunct*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ispunct*() and *ispunct_l*() functions shall test whether *c* is a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *ispunct_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *ispunct*() and *ispunct_l*() functions shall return non-zero if *c* is a punctuation character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *isspace*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isspace, isspace_l — test for a white-space character

SYNOPSIS

```
#include <ctype.h>

int isspace(int c);
int isspace_l(int c, locale_t locale);
```

DESCRIPTION

For *isspace*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isspace*() and *isspace_l*() functions shall test whether *c* is a character of class **space** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isspace_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isspace*() and *isspace_l*() functions shall return non-zero if *c* is a white-space character; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isupper*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isunordered — test if arguments are unordered

SYNOPSIS

#include <math.h>

int isunordered(real-floating *x*, real-floating *y*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isunordered*() macro shall determine whether its arguments are unordered.

RETURN VALUE

Upon successful completion, the *isunordered*() macro shall return 1 if its arguments are unordered, and 0 otherwise.

If *x* or *y* is NaN, 1 shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The relational and equality operators support the usual mathematical relationships between numeric values. For any ordered pair of numeric values, exactly one of the relationships (less, greater, and equal) is true. Relational operators may raise the invalid floating-point exception when argument values are NaNs. For a NaN and a numeric value, or for two NaNs, just the unordered relationship is true. This macro is a quiet (non-floating-point exception raising) version of a relational operator. It facilitates writing efficient code that accounts for NaNs without suffering the invalid floating-point exception. In the SYNOPSIS section, **real-floating** indicates that the argument shall be an expression of **real-floating** type.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isgreater( ), *isgreaterequal*( ), *isless*( ), *islessequal*( ), *islessgreater*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isupper, isupper_l — test for an uppercase letter

SYNOPSIS

```
#include <ctype.h>

int isupper(int c);
int isupper_l(int c, locale_t locale);
```

DESCRIPTION

For *isupper*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isupper*() and *isupper_l*() functions shall test whether *c* is a character of class **upper** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isupper_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isupper*() and *isupper_l*() functions shall return non-zero if *c* is an uppercase letter; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswalnum, iswalnum_l — test for an alphanumeric wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswalnum(wint_t wc);
int iswalnum_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswalnum*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswalnum*() and *iswalnum_l*() functions shall test whether *wc* is a wide-character code representing a character of class **alpha** or **digit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswalnum_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswalnum*() and *iswalnum_l*() functions shall return non-zero if *wc* is an alphanumeric wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalpha( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<stdio.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswalpha, iswalpha_l — test for an alphabetic wide-character code

SYNOPSIS

#include <wctype.h>

int iswalpha(wint_t *wc*);
int iswalpha_l(wint_t *wc*, locale_t *locale*);

DESCRIPTION

For *iswalpha*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswalpha*() and *iswalpha_l*() functions shall test whether *wc* is a wide-character code representing a character of class **alpha** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswalpha_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswalpha*() and *iswalpha_l*() functions shall return non-zero if *wc* is an alphabetic wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswblank, iswblank_l — test for a blank wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswblank(wint_t wc);
int iswblank_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswblank*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswblank*() and *iswblank*() functions shall test whether *wc* is a wide-character code representing a character of class **blank** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswblank_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswblank*() and *iswblank_l*() functions shall return non-zero if *wc* is a blank wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswcntrl, iswcntrl_l — test for a control wide-character code

SYNOPSIS

#include <wctype.h>

int iswcntrl(wint_t *wc*);
int iswcntrl_l(wint_t *wc*, locale_t *locale*);

DESCRIPTION

For *iswcntrl*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswcntrl*() and *iswcntrl_l*() functions shall test whether *wc* is a wide-character code representing a character of class **cntrl** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswcntrl_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswcntrl*() and *iswcntrl_l*() functions shall return non-zero if *wc* is a control wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswctype, iswctype_l — test character for a specified class

SYNOPSIS

```
#include <wctype.h>

int iswctype(wint_t wc, wctype_t charclass);
int iswctype_l(wint_t wc, wctype_t charclass,
    locale_t locale);
```

DESCRIPTION

For *iswctype*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswctype*() and *iswctype_l*() functions shall determine whether the wide-character code *wc* has the character class *charclass*, returning true or false. The *iswctype*() and *iswctype_l*() functions are defined on WEOF and wide-character codes corresponding to the valid character encodings in the current locale, or in the locale represented by *locale*, respectively. If the *wc* argument is not in the domain of the function, the result is undefined. If the value of *charclass* is invalid (that is, not obtained by a call to *wctype*() or *charclass* is invalidated by a subsequent call to *setlocale*() that has affected category *LC_CTYPE*) the result is unspecified.

The behavior is undefined if the *locale* argument to *iswctype_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswctype*() and *iswctype_l*() functions shall return non-zero (true) if and only if *wc* has the property described by *charclass*. If *charclass* is (**wctype_t**)0, these functions shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Testing for a Valid Character

```
#include <wctype.h>
...
int yes_or_no;
wint_t wc;
wctype_t valid_class;
...
if ((valid_class=wctype("vowel")) == (wctype_t)0)
    /* Invalid character class. */
yes_or_no=iswctype(wc,valid_class);
```

APPLICATION USAGE

The twelve strings **"alnum"**, **"alpha"**, **"blank"**, **"cntrl"**, **"digit"**, **"graph"**, **"lower"**, **"print"**, **"punct"**, **"space"**, **"upper"**, and **"xdigit"** are reserved for the standard character classes. In the table below, the functions in the left column are equivalent to the functions in the right column.

iswalnum(*wc*)	iswctype(*wc*, wctype("alnum"))
iswalnum_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("alnum"), *locale*)
iswalpha(*wc*)	iswctype(*wc*, wctype("alpha"))

iswalpha_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("alpha"), *locale*)
iswblank(*wc*)	iswctype(*wc*, wctype("blank"))
iswblank_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("blank"), *locale*)
iswcntrl(*wc*)	iswctype(*wc*, wctype("cntrl"))
iswcntrl_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("cntrl"), *locale*)
iswdigit(*wc*)	iswctype(*wc*, wctype("digit"))
iswdigit_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("digit"), *locale*)
iswgraph(*wc*)	iswctype(*wc*, wctype("graph"))
iswgraph_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("graph"), *locale*)
iswlower(*wc*)	iswctype(*wc*, wctype("lower"))
iswlower_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("lower"), *locale*)
iswprint(*wc*)	iswctype(*wc*, wctype("print"))
iswprint_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("print"), *locale*)
iswpunct(*wc*)	iswctype(*wc*, wctype("punct"))
iswpunct_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("punct"), *locale*)
iswspace(*wc*)	iswctype(*wc*, wctype("space"))
iswspace_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("space"), *locale*)
iswupper(*wc*)	iswctype(*wc*, wctype("upper"))
iswupper_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("upper"), *locale*)
iswxdigit(*wc*)	iswctype(*wc*, wctype("xdigit"))
iswxdigit_l(*wc*, *locale*)	iswctype_l(*wc*, wctype("xdigit"), *locale*)

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *isws-pace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( ), *wctype*( )

The Base Definitions volume of POSIX.1-2017, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswdigit, iswdigit_l — test for a decimal digit wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswdigit(wint_t wc);
int iswdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswdigit*() and *iswdigit_l*() functions shall test whether *wc* is a wide-character code representing a character of class **digit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswdigit*() and *iswdigit_l*() functions shall return non-zero if *wc* is a decimal digit wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswgraph, iswgraph_l — test for a visible wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswgraph(wint_t wc);
int iswgraph_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswgraph*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswgraph*() and *iswgraph_l*() functions shall test whether *wc* is a wide-character code representing a character of class **graph** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswgraph_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswgraph*() and *iswgraph_l*() functions shall return non-zero if *wc* is a wide-character code with a visible representation; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswlower, iswlower_l — test for a lowercase letter wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswlower(wint_t wc);
int iswlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswlower*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswlower*() and *iswlower_l*() functions shall test whether *wc* is a wide-character code representing a character of class **lower** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswlower_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswlower*() and *iswlower_l*() functions shall return non-zero if *wc* is a lowercase letter wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )1

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswprint, iswprint_l — test for a printable wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswprint(wint_t wc);
int iswprint_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswprint*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswprint*() and *iswprint_l*() functions shall test whether *wc* is a wide-character code representing a character of class **print** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswprint_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswprint*() and *iswprint_l*() functions shall return non-zero if *wc* is a printable wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswpunct, iswpunct_l — test for a punctuation wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswpunct(wint_t wc);
int iswpunct_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswpunct*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswpunct*() and *iswpunct_l*() functions shall test whether *wc* is a wide-character code representing a character of class **punct** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswpunct_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswpunct*() and *iswpunct_l*() functions shall return non-zero if *wc* is a punctuation wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswspace, iswspace_l — test for a white-space wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswspace(wint_t wc);
int iswspace_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswspace*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswspace*() and *iswspace_l*() functions shall test whether *wc* is a wide-character code representing a character of class **space** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswspace_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswspace*() and *iswspace_l*() functions shall return non-zero if *wc* is a white-space wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswupper*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswupper, iswupper_l — test for an uppercase letter wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswupper(wint_t wc);
int iswupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswupper*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswupper*() and *iswupper_l*() functions shall test whether *wc* is a wide-character code representing a character of class **upper** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswupper_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswupper*() and *iswupper_l*() functions shall return non-zero if *wc* is an uppercase letter wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswxdigit*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

iswxdigit, iswxdigit_l — test for a hexadecimal digit wide-character code

SYNOPSIS

```
#include <wctype.h>

int iswxdigit(wint_t wc);
int iswxdigit_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *iswxdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *iswxdigit*() and *iswxdigit_l*() functions shall test whether *wc* is a wide-character code representing a character of class **xdigit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *wc* argument is a **wint_t**, the value of which the application shall ensure is a wide-character code corresponding to a valid character in the locale used by the function, or equal to the value of the macro WEOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *iswxdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *iswxdigit*() and *iswxdigit_l*() functions shall return non-zero if *wc* is a hexadecimal digit wide-character code; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswalnum( ), *iswalpha*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *setlocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

isxdigit, isxdigit_l — test for a hexadecimal digit

SYNOPSIS

```
#include <ctype.h>

int isxdigit(int c);
int isxdigit_l(int c, locale_t locale);
```

DESCRIPTION

For *isxdigit*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *isxdigit*() and *isxdigit_l*() functions shall test whether *c* is a character of class **xdigit** in the current locale, or in the locale represented by *locale*, respectively; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*.

The *c* argument is an **int**, the value of which the application shall ensure is a character representable as an **unsigned char** or equal to the value of the macro EOF. If the argument has any other value, the behavior is undefined.

The behavior is undefined if the *locale* argument to *isxdigit_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *isxdigit*() and *isxdigit_l*() functions shall return non-zero if *c* is a hexadecimal digit; otherwise, they shall return 0.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

To ensure applications portability, especially across natural languages, only these functions and the functions in the reference pages listed in the SEE ALSO section should be used for character classification.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

isalnum( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

j0, j1, jn — Bessel functions of the first kind

SYNOPSIS

```
#include <math.h>

double j0(double x);
double j1(double x);
double jn(int n, double x);
```

DESCRIPTION

The *j0*(), *j1*(), and *jn*() functions shall compute Bessel functions of *x* of the first kind of orders 0, 1, and *n*, respectively.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the relevant Bessel value of *x* of the first kind.

If the *x* argument is too large in magnitude, or the correct result would cause underflow, 0 shall be returned and a range error may occur.

If *x* is NaN, a NaN shall be returned.

ERRORS

These functions may fail if:

Range Error The value of *x* was too large in magnitude, or an underflow occurred.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

No other errors shall occur.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *y0*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

jrand48 — generate a uniformly distributed pseudo-random long signed integer

SYNOPSIS

```
#include <stdlib.h>

long jrand48(unsigned short xsubi[3]);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

kill — send a signal to a process or a group of processes

SYNOPSIS

#include <signal.h>

int kill(pid_t *pid*, int *sig*);

DESCRIPTION

The *kill*() function shall send a signal to a process or a group of processes specified by *pid*. The signal to be sent is specified by *sig* and is either one from the list given in *<signal.h>* or 0. If *sig* is 0 (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

For a process to have permission to send a signal to a process designated by *pid*, unless the sending process has appropriate privileges, the real or effective user ID of the sending process shall match the real or saved set-user-ID of the receiving process.

If *pid* is greater than 0, *sig* shall be sent to the process whose process ID is equal to *pid*.

If *pid* is 0, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.

If *pid* is −1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) for which the process has permission to send that signal.

If *pid* is negative, but not −1, *sig* shall be sent to all processes (excluding an unspecified set of system processes) whose process group ID is equal to the absolute value of *pid*, and for which the process has permission to send a signal.

If the value of *pid* causes *sig* to be generated for the sending process, and if *sig* is not blocked for the calling thread and if no other thread has *sig* unblocked or is waiting in a *sigwait*() function for *sig*, either *sig* or at least one pending unblocked signal shall be delivered to the sending thread before *kill*() returns.

The user ID tests described above shall not be applied when sending SIGCONT to a process that is a member of the same session as the sending process.

An implementation that provides extended security controls may impose further implementation-defined restrictions on the sending of signals, including the null signal. In particular, the system may deny the existence of some or all of the processes specified by *pid*.

The *kill*() function is successful if the process has permission to send *sig* to any of the processes specified by *pid*. If *kill*() fails, no signal shall be sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *kill*() function shall fail if:

EINVAL
The value of the *sig* argument is an invalid or unsupported signal number.

EPERM
The process does not have permission to send the signal to any receiving process.

ESRCH
No process or process group can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The semantics for permission checking for *kill*() differed between System V and most other implementations, such as Version 7 or 4.3 BSD. The semantics chosen for this volume of POSIX.1-2017 agree with System V. Specifically, a set-user-ID process cannot protect itself against signals (or at least not against SIGKILL) unless it changes its real user ID. This choice allows the user who starts an application to send it signals even if it changes its effective user ID. The other semantics give more power to an application that wants to protect itself from the user who ran it.

Some implementations provide semantic extensions to the *kill*() function when the absolute value of *pid* is greater than some maximum, or otherwise special, value. Negative values are a flag to *kill*(). Since most implementations return **[ESRCH]** in this case, this behavior is not included in this volume of POSIX.1-2017, although a conforming implementation could provide such an extension.

The unspecified processes to which a signal cannot be sent may include the scheduler or *init*.

There was initially strong sentiment to specify that, if *pid* specifies that a signal be sent to the calling process and that signal is not blocked, that signal would be delivered before *kill*() returns. This would permit a process to call *kill*() and be guaranteed that the call never return. However, historical implementations that provide only the *signal*() function make only the weaker guarantee in this volume of POSIX.1-2017, because they only deliver one signal each time a process enters the kernel. Modifications to such implementations to support the *sigaction*() function generally require entry to the kernel following return from a signal-catching function, in order to restore the signal mask. Such modifications have the effect of satisfying the stronger requirement, at least when *sigaction*() is used, but not necessarily when *signal*() is used. The standard developers considered making the stronger requirement except when *signal*() is used, but felt this would be unnecessarily complex. Implementors are encouraged to meet the stronger requirement whenever possible. In practice, the weaker requirement is the same, except in the rare case when two signals arrive during a very short window. This reasoning also applies to a similar requirement for *sigprocmask*().

In 4.2 BSD, the SIGCONT signal can be sent to any descendant process regardless of user-ID security checks. This allows a job control shell to continue a job even if processes in the job have altered their user IDs (as in the *su* command). In keeping with the addition of the concept of sessions, similar functionality is provided by allowing the SIGCONT signal to be sent to any process in the same session regardless of user ID security checks. This is less restrictive than BSD in the sense that ancestor processes (in the same session) can now be the recipient. It is more restrictive than BSD in the sense that descendant processes that form new sessions are now subject to the user ID checks. A similar relaxation of security is not necessary for the other job control signals since those signals are typically sent by the terminal driver in recognition of special characters being typed; the terminal driver bypasses all security checks.

In secure implementations, a process may be restricted from sending a signal to a process having a different security label. In order to prevent the existence or nonexistence of a process from being used as a covert channel, such processes should appear nonexistent to the sender; that is, **[ESRCH]** should be returned, rather than **[EPERM]**, if *pid* refers only to such processes.

Historical implementations varied on the result of a *kill*() with *pid* indicating a zombie process. Some indicated success on such a call (subject to permission checking), while others gave an error of **[ESRCH]**. Since the definition of process lifetime in this volume of POSIX.1-2017 covers zombie processes, the **[ESRCH]** error as described is inappropriate in this case and implementations that give this error do not conform. This means that an application cannot have a parent process check for termination of a particular child by sending it the null signal with *kill*(), but must instead use *waitpid*() or *waitid*().

There is some belief that the name *kill*() is misleading, since the function is not always intended to cause process termination. However, the name is common to all historical implementations, and any change would be in conflict with the goal of minimal changes to existing application code.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *raise*( ), *setsid*( ), *sigaction*( ), *sigqueue*( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

killpg — send a signal to a process group

SYNOPSIS

#include <signal.h>

int killpg(pid_t *pgrp*, int *sig*);

DESCRIPTION

The *killpg*() function shall send the signal specified by *sig* to the process group specified by *pgrp*.

If *pgrp* is greater than 1, *killpg*(*pgrp*, *sig*) shall be equivalent to *kill*(−*pgrp*, *sig*). If *pgrp* is less than or equal to 1, the behavior of *killpg*() is undefined.

RETURN VALUE

Refer to *kill*( ).

ERRORS

Refer to *kill*( ).

The following sections are informative.

EXAMPLES

Sending a Signal to All Other Members of a Process Group

The following example shows how the calling process could send a signal to all other members of its process group. To prevent itself from receiving the signal it first makes itself immune to the signal by ignoring it.

```
#include <signal.h>
#include <unistd.h>
...
  if (signal(SIGUSR1, SIG_IGN) == SIG_ERR)
     /* Handle error */;

  if (killpg(getpgrp(), SIGUSR1) == -1)
     /* Handle error */;"
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpgid( ), *getpid*( ), *kill*( ), *raise*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

l64a — convert a 32-bit integer to a radix-64 ASCII string

SYNOPSIS

```
#include <stdlib.h>

char *l64a(long value);
```

DESCRIPTION

Refer to *a64l*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

labs, llabs — return a long integer absolute value

SYNOPSIS

```
#include <stdlib.h>

long labs(long i);
long long llabs(long long i);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *labs*() function shall compute the absolute value of the **long** integer operand *i*. The *llabs*() function shall compute the absolute value of the **long long** integer operand *i*. If the result cannot be represented, the behavior is undefined.

RETURN VALUE

The *labs*() function shall return the absolute value of the **long** integer operand.

The *llabs*() function shall return the absolute value of the **long long** integer operand.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lchown — change the owner and group of a symbolic link

SYNOPSIS

#include <unistd.h>

int lchown(const char **path*, uid_t *owner*, gid_t *group*);

DESCRIPTION

The *lchown*() function shall be equivalent to *chown*(), except in the case where the named file is a symbolic link. In this case, *lchown*() shall change the ownership of the symbolic link file itself, while *chown*() changes the ownership of the file or directory to which the symbolic link refers.

RETURN VALUE

Upon successful completion, *lchown*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate an error.

ERRORS

The *lchown*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix of *path*.

EINVAL
The owner or group ID is not a value supported by the implementation.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The effective user ID does not match the owner of the file and the process does not have appropriate privileges.

EROFS
The file resides on a read-only file system.

The *lchown*() function may fail if:

EIO An I/O error occurred while reading or writing to the file system.

EINTR
A signal was caught during execution of the function.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Changing the Current Owner of a File

The following example shows how to change the ownership of the symbolic link named **/modules/pass1** to the user ID associated with "jones" and the group ID associated with "cnd".

The numeric value for the user ID is obtained by using the *getpwnam*() function. The numeric value for the group ID is obtained by using the *getgrnam*() function.

```
#include <sys/types.h>
#include <unistd.h>
#include <pwd.h>
#include <grp.h>

struct passwd *pwd;
struct group  *grp;
char          *path = "/modules/pass1";
...
pwd = getpwnam("jones");
grp = getgrnam("cnd");
lchown(path, pwd->pw_uid, grp->gr_gid);
```

APPLICATION USAGE

On implementations which support symbolic links as directory entries rather than files, *lchown*() may fail.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

chown( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lcong48 — seed a uniformly distributed pseudo-random signed long integer generator

SYNOPSIS

#include <stdlib.h>

void lcong48(unsigned short *param*[7]);

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ldexp, ldexpf, ldexpl — load exponent of a floating-point number

SYNOPSIS

#include <math.h>

double ldexp(double *x*, int *exp*);
float ldexpf(float *x*, int *exp*);
long double ldexpl(long double *x*, int *exp*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the quantity $x * 2^{exp}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *x* multiplied by 2, raised to the power *exp*.

If these functions would cause overflow, a range error shall occur and *ldexp*(), *ldexpf*(), and *ldexpl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*), respectively.

If the correct value would cause underflow, and is not representable, a range error may occur, and *ldexp*(), *ldexpf*(), and *ldexpl*() shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *exp* is 0, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *frexp*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ldiv, lldiv — compute quotient and remainder of a long division

SYNOPSIS

```
#include <stdlib.h>

ldiv_t ldiv(long numer, long denom);
lldiv_t lldiv(long long numer, long long denom);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the quotient and remainder of the division of the numerator *numer* by the denominator *denom*. If the division is inexact, the resulting quotient is the **long** integer (for the *ldiv*() function) or **long long** integer (for the *lldiv*() function) of lesser magnitude that is the nearest to the algebraic quotient. If the result cannot be represented, the behavior is undefined; otherwise, *quot* * *denom*+*rem* shall equal *numer*.

RETURN VALUE

The *ldiv*() function shall return a structure of type **ldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long   quot;    /* Quotient */
long   rem;     /* Remainder */
```

The *lldiv*() function shall return a structure of type **lldiv_t**, comprising both the quotient and the remainder. The structure shall include the following members, in any order:

```
long long   quot;    /* Quotient */
long long   rem;     /* Remainder */
```

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

div( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lfind — find entry in a linear search table

SYNOPSIS

#include <search.h>

void *lfind(const void **key*, const void **base*, size_t **nelp*,
  size_t *width*, int (**compar*)(const void *, const void *));

DESCRIPTION

Refer to *lsearch*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

lgamma, lgammaf, lgammal, signgam — log gamma function

SYNOPSIS

#include <math.h>

double lgamma(double *x*);
float lgammaf(float *x*);
long double lgammal(long double *x*);
extern int signgam;

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute $log_ e" "|Γ (^ x)|$ where $Γ (^ x)$ is defined as $int from 0 to inf e"^" " "{ - t } t"^" " "{ x - 1 } dt$. The argument *x* need not be a non-positive integer ($Γ(^ x)$ is defined over the reals, except the non-positive integers).

If *x* is NaN, −Inf, or a negative integer, the value of *signgam* is unspecified.

These functions need not be thread-safe.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the logarithmic gamma of *x*.

If *x* is a non-positive integer, a pole error shall occur and *lgamma*(), *lgammaf*(), and *lgammal*() shall return +HUGE_VAL, +HUGE_VALF, and +HUGE_VALL, respectively.

If the correct value would cause overflow, a range error shall occur and *lgamma*(), *lgammaf*(), and *lgammal*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (having the same sign as the correct value), respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is 1 or 2, +0 shall be returned.

If *x* is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Pole Error The *x* argument is a negative integer or zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

link, linkat — link one file to another file

SYNOPSIS

```
#include <unistd.h>

int link(const char *path1, const char *path2);

#include <fcntl.h>

int linkat(int fd1, const char *path1, int fd2,
   const char *path2, int flag);
```

DESCRIPTION

The *link*() function shall create a new link (directory entry) for the existing file, *path1*.

The *path1* argument points to a pathname naming an existing file. The *path2* argument points to a pathname naming the new directory entry to be created. The *link*() function shall atomically create a new link for the existing file and the link count of the file shall be incremented by one.

If *path1* names a directory, *link*() shall fail unless the process has appropriate privileges and the implementation supports using *link*() on directories.

If *path1* names a symbolic link, it is implementation-defined whether *link*() follows the symbolic link, or creates a new link to the symbolic link itself.

Upon successful completion, *link*() shall mark for update the last file status change timestamp of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

If *link*() fails, no link shall be created and the link count of the file shall remain unchanged.

The implementation may require that the calling process has permission to access the existing file.

The *linkat*() function shall be equivalent to the *link*() function except that symbolic links shall be handled as specified by the value of *flag* (see below) and except in the case where either *path1* or *path2* or both are relative paths. In this case a relative path *path1* is interpreted relative to the directory associated with the file descriptor *fd1* instead of the current working directory and similarly for *path2* and the file descriptor *fd2*. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_SYMLINK_FOLLOW
: If *path1* names a symbolic link, a new link for the target of the symbolic link is created.

If *linkat*() is passed the special value AT_FDCWD in the *fd1* or *fd2* parameter, the current working directory shall be used for the respective *path* argument. If both *fd1* and *fd2* have value AT_FDCWD, the behavior shall be identical to a call to *link*(), except that symbolic links shall be handled as specified by the value of *flag*.

If the AT_SYMLINK_FOLLOW flag is clear in the *flag* argument and the *path1* argument names a symbolic link, a new link is created for the symbolic link *path1* and not its target.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EACCES
A component of either path prefix denies search permission, or the requested link requires writing in a directory that denies write permission, or the calling process does not have permission to access the existing file and this is required by the implementation.

EEXIST
The *path2* argument resolves to an existing directory entry or refers to a symbolic link.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path1* or *path2* argument.

EMLINK
The number of links to the file named by *path1* would exceed {LINK_MAX}.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of either path prefix does not exist; the file named by *path1* does not exist; or *path1* or *path2* points to an empty string.

ENOENT or **ENOTDIR**
The *path1* argument names an existing non-directory file, and the *path2* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path2* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory to contain the link cannot be extended.

ENOTDIR
A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory, or the *path1* argument names an existing non-directory file and the *path2* argument names a nonexistent file, contains at least one non-<slash> character, and ends with one or more trailing <slash> characters.

EPERM
The file named by *path1* is a directory and either the calling process does not have appropriate privileges or the implementation prohibits using *link*() on directories.

EROFS
The requested link requires writing in a directory on a read-only file system.

EXDEV
The link named by *path2* and the file named by *path1* are on different file systems and the implementation does not support links between file systems.

EXDEV
path1 refers to a named STREAM.

The *linkat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd1* or *fd2* is not O_SEARCH and the permissions of the directory underlying *fd1* or *fd2*, respectively, do not permit directory searches.

EBADF
The *path1* or *path2* argument does not specify an absolute path and the *fd1* or *fd2* argument, respectively, is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path1* or *path2* argument is not an absolute path and *fd1* or *fd2*, respectively, is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path1* or *path2* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The *linkat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

Creating a Link to a File

The following example shows how to create a link to a file named **/home/cnd/mod1** by creating a new directory entry named **/modules/pass1**.

```
#include <unistd.h>

char *path1 = "/home/cnd/mod1";
char *path2 = "/modules/pass1";
int   status;
...
status = link (path1, path2);
```

Creating a Link to a File Within a Program

In the following program example, the *link*() function links the **/etc/passwd** file (defined as **PASSWDFILE**) to a file named **/etc/opasswd** (defined as **SAVEFILE**), which is used to save the current password file. Then, after removing the current password file (defined as **PASSWDFILE**), the new password file is saved as the current password file using the *link*() function again.

```
#include <unistd.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* Save current password file */
link (PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink (PASSWDFILE);

/* Save new password file as current password file. */
link (LOCKFILE,PASSWDFILE);
```

APPLICATION USAGE

Some implementations do allow links between file systems.

If *path1* refers to a symbolic link, application developers should use *linkat*() with appropriate flags to select whether or not the symbolic link should be resolved.

RATIONALE

Linking to a directory is restricted to the superuser in most historical implementations because this capability may produce loops in the file hierarchy or otherwise corrupt the file system. This volume of POSIX.1-2017 continues that philosophy by prohibiting *link*() and *unlink*() from doing this. Other functions could do it if the implementor designed such an extension.

Some historical implementations allow linking of files on different file systems. Wording was added to explicitly allow this optional behavior.

The exception for cross-file system links is intended to apply only to links that are programmatically indistinguishable from "hard" links.

The purpose of the *linkat*() function is to link files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *link*(), resulting in unspecified behavior. By opening a file descriptor for the directory of both the existing file and the target location and using the *linkat*() function it can be guaranteed that the both filenames are in the desired directories.

The AT_SYMLINK_FOLLOW flag allows for implementing both common behaviors of the *link*() function. The POSIX specification requires that if *path1* is a symbolic link, a new link for the target of the symbolic link is created. Many systems by default or as an alternative provide a mechanism to avoid the implicit symbolic link lookup and create a new link for the symbolic link itself.

Earlier versions of this standard specified only the *link*() function, and required it to behave like *linkat*() with the AT_SYMLINK_FOLLOW flag. However, historical practice from SVR4 and Linux kernels had *link*() behaving like *linkat*() with no flags, and many systems that attempted to provide a conforming *link*() function did so in a way that was rarely used, and when it was used did not conform to the standard (e.g., by not being atomic, or by dereferencing the symbolic link incorrectly). Since applications could not rely on *link*() following links in practice, the *linkat*() function was added taking a flag to specify the desired behavior for the application.

FUTURE DIRECTIONS

None.

SEE ALSO

rename( ), *symlink*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lio_listio — list directed I/O

SYNOPSIS

```
#include <aio.h>

int lio_listio(int mode, struct aiocb *restrict const list[restrict],
   int nent, struct sigevent *restrict sig);
```

DESCRIPTION

The *lio_listio*() function shall initiate a list of I/O requests with a single function call.

The *mode* argument takes one of the values LIO_WAIT or LIO_NOWAIT declared in <*aio.h*> and determines whether the function returns when the I/O operations have been completed, or as soon as the operations have been queued. If the *mode* argument is LIO_WAIT, the function shall wait until all I/O is complete and the *sig* argument shall be ignored.

If the *mode* argument is LIO_NOWAIT, the function shall return immediately, and asynchronous notification shall occur, according to the *sig* argument, when all the I/O operations complete. If *sig* is NULL, then no asynchronous notification shall occur. If *sig* is not NULL, asynchronous notification occurs as specified in *Section 2.4.1*, *Signal Generation and Delivery* when all the requests in *list* have completed.

The I/O requests enumerated by *list* are submitted in an unspecified order.

The *list* argument is an array of pointers to **aiocb** structures. The array contains *nent* elements. The array may contain NULL elements, which shall be ignored.

If the buffer pointed to by *list* or the **aiocb** structures pointed to by the elements of the array *list* become illegal addresses before all asynchronous I/O completed and, if necessary, the notification is sent, then the behavior is undefined. If the buffers pointed to by the *aio_buf* member of the **aiocb** structure pointed to by the elements of the array *list* become illegal addresses prior to the asynchronous I/O associated with that **aiocb** structure being completed, the behavior is undefined.

The *aio_lio_opcode* field of each **aiocb** structure specifies the operation to be performed. The supported operations are LIO_READ, LIO_WRITE, and LIO_NOP; these symbols are defined in <*aio.h*>. The LIO_NOP operation causes the list entry to be ignored. If the *aio_lio_opcode* element is equal to LIO_READ, then an I/O operation is submitted as if by a call to *aio_read*() with the *aiocbp* equal to the address of the **aiocb** structure. If the *aio_lio_opcode* element is equal to LIO_WRITE, then an I/O operation is submitted as if by a call to *aio_write*() with the *aiocbp* equal to the address of the **aiocb** structure.

The *aio_fildes* member specifies the file descriptor on which the operation is to be performed.

The *aio_buf* member specifies the address of the buffer to or from which the data is transferred.

The *aio_nbytes* member specifies the number of bytes of data to be transferred.

The members of the **aiocb** structure further describe the I/O operation to be performed, in a manner identical to that of the corresponding **aiocb** structure when used by the *aio_read*() and *aio_write*() functions.

The *nent* argument specifies how many elements are members of the list; that is, the length of the array.

The behavior of this function is altered according to the definitions of synchronized I/O data integrity completion and synchronized I/O file integrity completion if synchronized I/O is enabled on the file associated with *aio_fildes*.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *aiocbp*−>*aio_fildes*.

If *sig*−>*sigev_notify* is SIGEV_THREAD and *sig*−>*sigev_notify_attributes* is a non-null pointer and the block pointed to by this pointer becomes an illegal address prior to all asynchronous I/O being completed, then the behavior is undefined.

RETURN VALUE

If the *mode* argument has the value LIO_NOWAIT, the *lio_listio*() function shall return the value zero if the I/O operations are successfully queued; otherwise, the function shall return the value −1 and set *errno* to indicate the error.

If the *mode* argument has the value LIO_WAIT, the *lio_listio*() function shall return the value zero when all the indicated I/O has completed successfully. Otherwise, *lio_listio*() shall return a value of −1 and set *errno* to indicate the error.

In either case, the return value only indicates the success or failure of the *lio_listio*() call itself, not the status of the individual I/O requests. In some cases one or more of the I/O requests contained in the list may fail. Failure of an individual request does not prevent completion of any other individual request. To determine the outcome of each I/O request, the application shall examine the error status associated with each **aiocb** control block. The error statuses so returned are identical to those returned as the result of an *aio_read*() or *aio_write*() function.

ERRORS

The *lio_listio*() function shall fail if:

EAGAIN
The resources necessary to queue all the I/O requests were not available. The application may check the error status for each **aiocb** to determine the individual request(s) that failed.

EAGAIN
The number of entries indicated by *nent* would cause the system-wide limit {AIO_MAX} to be exceeded.

EINVAL
The *mode* argument is not a proper value, or the value of *nent* was greater than {AIO_LISTIO_MAX}.

EINTR
A signal was delivered while waiting for all I/O requests to complete during an LIO_WAIT operation. Note that, since each I/O operation invoked by *lio_listio*() may possibly provoke a signal when it completes, this error return may be caused by the completion of one (or more) of the very I/O operations being awaited. Outstanding I/O requests are not canceled, and the application shall examine each list element to determine whether the request was initiated, canceled, or completed.

EIO One or more of the individual I/O operations failed. The application may check the error status for each **aiocb** structure to determine the individual request(s) that failed.

In addition to the errors returned by the *lio_listio*() function, if the *lio_listio*() function succeeds or fails with errors of **[EAGAIN]**, **[EINTR]**, or **[EIO]**, then some of the I/O specified by the list may have been initiated. If the *lio_listio*() function fails with an error code other than **[EAGAIN]**, **[EINTR]**, or **[EIO]**, no operations from the list shall have been initiated. The I/O operation indicated by each list element can encounter errors specific to the individual read or write function being performed. In this event, the error status for each **aiocb** control block contains the associated error code. The error codes that can be set are the same as would be set by a *read*() or *write*() function, with the following additional error codes possible:

EAGAIN
The requested I/O operation was not queued due to resource limitations.

ECANCELED
The requested I/O was canceled before the I/O completed due to an explicit *aio_cancel*() request.

EFBIG
The *aiocbp*−>*aio_lio_opcode* is LIO_WRITE, the file is a regular file, *aiocbp*−>*aio_nbytes* is greater than 0, and the *aiocbp*−>*aio_offset* is greater than or equal to the offset maximum in the open file description associated with *aiocbp*−>*aio_fildes*.

EINPROGRESS
The requested I/O is in progress.

EOVERFLOW
The *aiocbp*−>*aio_lio_opcode* is LIO_READ, the file is a regular file, *aiocbp*−>*aio_nbytes* is greater than 0, and the *aiocbp*−>*aio_offset* is before the end-of-file and is greater than or equal to the offset maximum in the open file description associated with *aiocbp*−>*aio_fildes*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Although it may appear that there are inconsistencies in the specified circumstances for error codes, the **[EIO]** error condition applies when any circumstance relating to an individual operation makes that operation fail. This might be due to a badly formulated request (for example, the *aio_lio_opcode* field is invalid, and *aio_error*() returns **[EINVAL]**) or might arise from application behavior (for example, the file descriptor is closed before the operation is initiated, and *aio_error*() returns **[EBADF]**).

The limitation on the set of error codes returned when operations from the list shall have been initiated enables applications to know when operations have been started and whether *aio_error*() is valid for a specific operation.

FUTURE DIRECTIONS

None.

SEE ALSO

aio_read( ), *aio_write*( ), *aio_error*( ), *aio_return*( ), *aio_cancel*( ), *close*( ), *exec*, *exit*( ), *fork*( ), *lseek*( ), *read*( )

The Base Definitions volume of POSIX.1-2017, **<aio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

listen — listen for socket connections and limit the queue of incoming connections

SYNOPSIS

#include <sys/socket.h>

int listen(int *socket*, int *backlog*);

DESCRIPTION

The *listen*() function shall mark a connection-mode socket, specified by the *socket* argument, as accepting connections.

The *backlog* argument provides a hint to the implementation which the implementation shall use to limit the number of outstanding connections in the socket's listen queue. Implementations may impose a limit on *backlog* and silently reduce the specified value. Normally, a larger *backlog* argument value shall result in a larger or equal length of the listen queue. Implementations shall support values of *backlog* up to SOMAXCONN, defined in *<sys/socket.h>*.

The implementation may include incomplete connections in its listen queue. The limits on the number of incomplete connections and completed connections queued may be different.

The implementation may have an upper limit on the length of the listen queue—either global or per accepting socket. If *backlog* exceeds this limit, the length of the listen queue is set to the limit.

If *listen*() is called with a *backlog* argument value that is less than 0, the function behaves as if it had been called with a *backlog* argument value of 0.

A *backlog* argument of 0 may allow the socket to accept connections, in which case the length of the listen queue may be set to an implementation-defined minimum value.

The socket in use may require the process to have appropriate privileges to use the *listen*() function.

RETURN VALUE

Upon successful completions, *listen*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *listen*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

EDESTADDRREQ
The socket is not bound to a local address, and the protocol does not support listening on an unbound socket.

EINVAL
The *socket* is already connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The socket protocol does not support *listen*().

The *listen*() function may fail if:

EACCES
The calling process does not have appropriate privileges.

EINVAL
The *socket* has been shut down.

ENOBUFS
Insufficient resources are available in the system to complete the call.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *connect*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer’s Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

llabs — return a long integer absolute value

SYNOPSIS

#include <stdlib.h>

long long llabs(long long *i*);

DESCRIPTION

Refer to *labs*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lldiv — compute quotient and remainder of a long division

SYNOPSIS

#include <stdlib.h>

lldiv_t lldiv(long long *numer*, long long *denom*);

DESCRIPTION

Refer to *ldiv*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

llrint, llrintf, llrintl — round to the nearest integer value using current rounding direction

SYNOPSIS

#include <math.h>

long long llrint(double *x*);
long long llrintf(float *x*);
long long llrintl(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is −Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error

The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *lrint*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

llround, llroundf, llroundl — round to nearest integer value

SYNOPSIS

#include <math.h>

long long llround(double *x*);
long long llroundf(float *x*);
long long llroundl(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If *x* is NaN, a domain error shall occur, and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is −Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error
: The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error
: The correct value is not representable as an integer.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *llrint*() functions in that the default rounding direction for the *llround*() functions round halfway cases away from zero and need not raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *lround*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

localeconv — return locale-specific information

SYNOPSIS

```
#include <locale.h>

struct lconv *localeconv(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *localeconv*() function shall set the components of an object with the type **struct lconv** with the values appropriate for the formatting of numeric quantities (monetary and otherwise) according to the rules of the current locale.

The members of the structure with type **char *** are pointers to strings, any of which (except **decimal_point**) can point to "", to indicate that the value is not available in the current locale or is of zero length. The members with type **char** are non-negative numbers, any of which can be {CHAR_MAX} to indicate that the value is not available in the current locale.

The members include the following:

char *decimal_point
The radix character used to format non-monetary quantities.

char *thousands_sep
The character used to separate groups of digits before the decimal-point character in formatted non-monetary quantities.

char *grouping
A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted non-monetary quantities.

char *int_curr_symbol
The international currency symbol applicable to the current locale. The first three characters contain the alphabetic international currency symbol in accordance with those specified in the ISO 4217: 2001 standard. The fourth character (immediately preceding the null byte) is the character used to separate the international currency symbol from the monetary quantity.

char *currency_symbol
The local currency symbol applicable to the current locale.

char *mon_decimal_point
The radix character used to format monetary quantities.

char *mon_thousands_sep
The separator for groups of digits before the decimal-point in formatted monetary quantities.

char *mon_grouping
A string whose elements taken as one-byte integer values indicate the size of each group of digits in formatted monetary quantities.

char *positive_sign
The string used to indicate a non-negative valued formatted monetary quantity.

char *negative_sign
The string used to indicate a negative valued formatted monetary quantity.

char int_frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in an internationally formatted monetary quantity.

char frac_digits
The number of fractional digits (those after the decimal-point) to be displayed in a formatted monetary quantity.

char p_cs_precedes
Set to 1 if the **currency_symbol** precedes the value for a non-negative formatted monetary quantity. Set to 0 if the symbol succeeds the value.

char p_sep_by_space
Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a non-negative formatted monetary quantity.

char n_cs_precedes
Set to 1 if the **currency_symbol** precedes the value for a negative formatted monetary quantity. Set to 0 if the symbol succeeds the value.

char n_sep_by_space
Set to a value indicating the separation of the **currency_symbol**, the sign string, and the value for a negative formatted monetary quantity.

char p_sign_posn
Set to a value indicating the positioning of the **positive_sign** for a non-negative formatted monetary quantity.

char n_sign_posn
Set to a value indicating the positioning of the **negative_sign** for a negative formatted monetary quantity.

char int_p_cs_precedes
Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a non-negative internationally formatted monetary quantity.

char int_n_cs_precedes
Set to 1 or 0 if the **int_curr_symbol** respectively precedes or succeeds the value for a negative internationally formatted monetary quantity.

char int_p_sep_by_space
Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a non-negative internationally formatted monetary quantity.

char int_n_sep_by_space
Set to a value indicating the separation of the **int_curr_symbol**, the sign string, and the value for a negative internationally formatted monetary quantity.

char int_p_sign_posn
Set to a value indicating the positioning of the **positive_sign** for a non-negative internationally formatted monetary quantity.

char int_n_sign_posn
Set to a value indicating the positioning of the **negative_sign** for a negative internationally formatted monetary quantity.

The elements of **grouping** and **mon_grouping** are interpreted according to the following:

{CHAR_MAX}	No further grouping is to be performed.
0	The previous element is to be repeatedly used for the remainder of the digits.
other	The integer value is the number of digits that comprise the current group. The next element is examined to determine the size of the next group of digits before the current group.

The values of **p_sep_by_space**, **n_sep_by_space**, **int_p_sep_by_space**, and **int_n_sep_by_space** are interpreted according to the following:

0 No space separates the currency symbol and value.

1 If the currency symbol and sign string are adjacent, a space separates them from the value; otherwise, a space separates the currency symbol from the value.

2 If the currency symbol and sign string are adjacent, a space separates them; otherwise, a space separates the sign string from the value.

For **int_p_sep_by_space** and **int_n_sep_by_space**, the fourth character of **int_curr_symbol** is used instead of a space.

The values of **p_sign_posn**, **n_sign_posn**, **int_p_sign_posn**, and **int_n_sign_posn** are interpreted according to the following:

0 Parentheses surround the quantity and **currency_symbol** or **int_curr_symbol**.

1 The sign string precedes the quantity and **currency_symbol** or **int_curr_symbol**.

2 The sign string succeeds the quantity and **currency_symbol** or **int_curr_symbol**.

3 The sign string immediately precedes the **currency_symbol** or **int_curr_symbol**.

4 The sign string immediately succeeds the **currency_symbol** or **int_curr_symbol**.

The implementation shall behave as if no function in this volume of POSIX.1-2017 calls *localeconv*().

The *localeconv*() function need not be thread-safe.

RETURN VALUE

The *localeconv*() function shall return a pointer to the filled-in object. The application shall not modify the structure to which the return value points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *localeconv*(). In addition, the returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by subsequent calls to *setlocale*() with the categories LC_ALL, LC_MONETARY, or LC_NUMERIC, or by calls to *uselocale*() which change the categories LC_MONETARY or LC_NUMERIC. The returned pointer, pointers within the structure, the structure, and the storage areas might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The following table illustrates the rules which may be used by four countries to format monetary quantities.

center box tab(!); cB | cB | cB | cB l | l | l | l. Country!Positive Format!Negative Format!International Format _ Italy!€.1.230!−€.1.230!EUR.1.230 Netherlands!€ 1.234,56!€ −1.234,56!EUR 1.234,56 Norway!kr1.234,56!kr1.234,56−!NOK 1.234,56 Switzerland!SFrs.1,234.56!SFrs.1,234.56C!CHF 1,234.56

For these four countries, the respective values for the monetary members of the structure returned by *localeconv*() are:

center box tab(!); cB | cB | cB | cB | cB lb | cf5 | cf5 | cf5 | cf5. !Italy!Netherlands!Norway!Switzerland _ int_curr_symbol!"EUR."!"EUR "!"NOK "!"CHF " currency_symbol!"€."!"€"!"kr"!"SFrs." mon_decimal_point!""!","!","!"." mon_thousands_sep!"."!"."!"."!"," mon_grouping!"\3"!"\3"!"\3"!"\3" positive_sign!""!""!""!"" negative_sign!"-"!"-"!"-"!"C" int_frac_digits!0!2!2!2 frac_digits!0!2!2!2 p_cs_precedes!1!1!1!1 p_sep_by_space!0!1!0!0 n_cs_precedes!1!1!1!1 n_sep_by_space!0!1!0!0 p_sign_posn!1!1!1!1 n_sign_posn!1!4!2!2 int_p_cs_precedes!1!1!1!1 int_n_cs_precedes!1!1!1!1

int_p_sep_by_space!0!0!0!0 int_n_sep_by_space!0!0!0!0 int_p_sign_posn!1!1!1!1
int_n_sign_posn!1!4!4!2

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fscanf*( ), *isalpha*( ), *isascii*( ), *nl_langinfo*( ), *setlocale*( ), *strcat*( ), *strchr*( ), *strcmp*( ), *strcoll*( ), *strcpy*( ), *strftime*( ), *strlen*( ), *strpbrk*( ), *strspn*( ), *strtok*( ), *strxfrm*( ), *strtod*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, **<langinfo.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

localtime, localtime_r — convert a time value to a broken-down local time

SYNOPSIS

```
#include <time.h>

struct tm *localtime(const time_t *timer);
struct tm *localtime_r(const time_t *restrict timer,
    struct tm *restrict result);
```

DESCRIPTION

For *localtime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *localtime*() function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time, expressed as a local time. The function corrects for the timezone and any seasonal time adjustments. Local timezone information is used as though *localtime*() calls *tzset*().

The relationship between a time in seconds since the Epoch used as an argument to *localtime*() and the **tm** structure (defined in the *<time.h>* header) is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*) corrected for timezone and any seasonal time adjustments, where the names in the structure and in the expression correspond.

The same relationship shall apply for *localtime_r*().

The *localtime*() function need not be thread-safe.

The *asctime*(), *ctime*(), *gmtime*(), and *localtime*() functions shall return values in one of two static objects: a broken-down time structure and an array of type **char**. Execution of any of the functions may overwrite the information returned in either of these objects by any of the other functions.

The *localtime_r*() function shall convert the time in seconds since the Epoch pointed to by *timer* into a broken-down time stored in the structure to which *result* points. The *localtime_r*() function shall also return a pointer to that same structure.

Unlike *localtime*(), the *localtime_r*() function is not required to set *tzname*. If *localtime_r*() sets *tzname*, it shall also set *daylight* and *timezone*. If *localtime_r*() does not set *tzname*, it shall not set *daylight* and shall not set *timezone*.

RETURN VALUE

Upon successful completion, the *localtime*() function shall return a pointer to the broken-down time structure. If an error is detected, *localtime*() shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, *localtime_r*() shall return a pointer to the structure pointed to by the argument *result*. If an error is detected, *localtime_r*() shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *localtime*() and *localtime_r*() functions shall fail if:

EOVERFLOW
The result cannot be represented.

The following sections are informative.

EXAMPLES

Getting the Local Date and Time

The following example uses the *time*() function to calculate the time elapsed, in seconds, since January 1, 1970 0:00 UTC (the Epoch), *localtime*() to convert that value to a broken-down time, and *asctime*() to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
    time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

Getting the Modification Time for a File

The following example prints the last data modification timestamp in the local timezone for a given file.

```
#include <stdio.h>
#include <time.h>
#include <sys/stat.h>

int
print_file_time(const char *pathname)
{
    struct stat statbuf;
    struct tm *tm;
    char timestr[BUFSIZ];

    if(stat(pathname, &statbuf) == -1)
        return -1;
    if((tm = localtime(&statbuf.st_mtime)) == NULL)
        return -1;
    if(strftime(timestr, sizeof(timestr), "%Y-%m-%d %H:%M:%S", tm) == 0)
        return -1;
    printf("%s: %s.%09ld\n", pathname, timestr, statbuf.st_mtim.tv_nsec);
    return 0;
}
```

Timing an Event

The following example gets the current time, converts it to a string using *localtime*() and *asctime*(), and prints it to standard output using *fputs*(). It then prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
```

```
time(&now);
printf("The time is ");
fputs(asctime(localtime(&now)), stdout);
printf("There are still %d minutes to the event.\n",
   minutes_to_event);
...
```

APPLICATION USAGE

The *localtime_r*() function is thread-safe and returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *time*( ), *tzset*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lockf — record locking on files

SYNOPSIS

#include <unistd.h>

int lockf(int *fildes*, int *function*, off_t *size*);

DESCRIPTION

The *lockf*() function shall lock sections of a file with advisory-mode locks. Calls to *lockf*() from threads in other processes which attempt to lock the locked file section shall either return an error value or block until the section becomes unlocked. All the locks for a process are removed when the process terminates. Record locking with *lockf*() shall be supported for regular files and may be supported for other files.

The *fildes* argument is an open file descriptor. To establish a lock with this function, the file descriptor shall be opened with write-only permission (O_WRONLY) or with read/write permission (O_RDWR).

The *function* argument is a control value which specifies the action to be taken. The permissible values for *function* are defined in *<unistd.h>* as follows:

box tab(!) center; cB | cB l | l. Function!Description _ F_ULOCK!Unlock locked sections. F_LOCK!Lock a section for exclusive use. F_TLOCK!Test and lock a section for exclusive use. F_TEST!Test a section for locks by other processes.

F_TEST shall detect if a lock by another process is present on the specified section.

F_LOCK and F_TLOCK shall both lock a section of a file if the section is available.

F_ULOCK shall remove locks from a section of the file.

The *size* argument is the number of contiguous bytes to be locked or unlocked. The section to be locked or unlocked starts at the current offset in the file and extends forward for a positive size or backward for a negative size (the preceding bytes up to but not including the current offset). If *size* is 0, the section from the current offset through the largest possible file offset shall be locked (that is, from the current offset through the present or any future end-of-file). An area need not be allocated to the file to be locked because locks may exist past the end-of-file.

The sections locked with F_LOCK or F_TLOCK may, in whole or in part, contain or be contained by a previously locked section for the same process. When this occurs, or if adjacent locked sections would occur, the sections shall be combined into a single locked section. If the request would cause the number of locks to exceed a system-imposed limit, the request shall fail.

F_LOCK and F_TLOCK requests differ only by the action taken if the section is not available. F_LOCK shall block the calling thread until the section is available. F_TLOCK shall cause the function to fail if the section is already locked by another process.

File locks shall be released on first close by the locking process of any file descriptor for the file.

F_ULOCK requests may release (wholly or in part) one or more locked sections controlled by the process. Locked sections shall be unlocked starting at the current file offset through *size* bytes or to the end-of-file if *size* is (**off_t**)0. When all of a locked section is not released (that is, when the beginning or end of the area to be unlocked falls within a locked section), the remaining portions of that section shall remain locked by the process. Releasing the center portion of a locked section shall cause the remaining locked beginning and end portions to become two separate locked sections. If the request would cause the number of locks in the system to exceed a system-imposed limit, the request shall fail.

A potential for deadlock occurs if the threads of a process controlling a locked section are blocked by accessing a locked section of another process. If the system detects that deadlock would occur, *lockf*() shall fail with an **[EDEADLK]** error.

The interaction between *fcntl*() and *lockf*() locks is unspecified.

Blocking on a section shall be interrupted by any signal.

An F_ULOCK request in which *size* is non-zero and the offset of the last byte of the requested section is the maximum value for an object of type **off_t**, when the process has an existing lock in which *size* is 0 and which includes the last byte of the requested section, shall be treated as a request to unlock from the start of the requested section with a size equal to 0. Otherwise, an F_ULOCK request shall attempt to unlock only the requested section.

Attempting to lock a section of a file that is associated with a buffered stream produces unspecified results.

RETURN VALUE

Upon successful completion, *lockf*() shall return 0. Otherwise, it shall return −1, set *errno* to indicate an error, and existing locks shall not be changed.

ERRORS

The *lockf*() function shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor; or *function* is F_LOCK or F_TLOCK and *fildes* is not a valid file descriptor open for writing.

EACCES or **EAGAIN**
The *function* argument is F_TLOCK or F_TEST and the section is already locked by another process.

EDEADLK
The *function* argument is F_LOCK and a deadlock is detected.

EINTR
A signal was caught during execution of the function.

EINVAL
The *function* argument is not one of F_LOCK, F_TLOCK, F_TEST, or F_ULOCK; or *size* plus the current file offset is less than 0.

EOVERFLOW
The offset of the first, or if *size* is not 0 then the last, byte in the requested section cannot be represented correctly in an object of type **off_t**.

The *lockf*() function may fail if:

EAGAIN
The *function* argument is F_LOCK or F_TLOCK and the file is mapped with *mmap*().

EDEADLK or **ENOLCK**
The *function* argument is F_LOCK, F_TLOCK, or F_ULOCK, and the request would cause the number of locks to exceed a system-imposed limit.

EOPNOTSUPP or **EINVAL**
The implementation does not support the locking of files of the type indicated by the *fildes* argument.

The following sections are informative.

EXAMPLES

Locking a Portion of a File

In the following example, a file named **/home/cnd/mod1** is being modified. Other processes that use locking are prevented from changing it during this process. Only the first 10 000 bytes are locked, and the lock call fails if another process has any part of this area locked already.

```
#include <fcntl.h>
#include <unistd.h>
```

```
int fildes;
int status;
...
fildes = open("/home/cnd/mod1", O_RDWR);
status = lockf(fildes, F_TLOCK, (off_t)10000);
```

APPLICATION USAGE

Record-locking should not be used in combination with the *fopen*(), *fread*(), *fwrite*(), and other *stdio* functions. Instead, the more primitive, non-buffered functions (such as *open*()) should be used. Unexpected results may occur in processes that do buffering in the user address space. The process may later read/write data which is/was locked. The *stdio* functions are the most common source of unexpected buffering.

The *alarm*() function may be used to provide a timeout facility in applications requiring it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *chmod*( ), *close*( ), *creat*( ), *fcntl*( ), *fopen*( ), *mmap*( ), *open*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

log, logf, logl — natural logarithm function

SYNOPSIS

```
#include <math.h>

double log(double x);
float logf(float x);
long double logl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the natural logarithm of their argument *x*, $\log_e(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of *x*.

If *x* is ±0, a pole error shall occur and *log*(), *logf*(), and *logl*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

For finite values of *x* that are less than 0, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is 1, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The finite value of *x* is negative, or *x* is −Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of *x* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *log10*( ), *log1p*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

log10, log10f, log10l — base 10 logarithm function

SYNOPSIS

#include <math.h>

double log10(double *x*);
float log10f(float *x*);
long double log10l(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the base 10 logarithm of their argument *x*, $\log_{10}(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 10 logarithm of *x*.

If *x* is ±0, a pole error shall occur and *log10*(), *log10f*(), and *log10l*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

For finite values of *x* that are less than 0, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is 1, +0 shall be returned.

If *x* is +Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Domain Error
: The finite value of *x* is negative, or *x* is −Inf.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error
: The value of *x* is zero.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *log*( ), *pow*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

log1p, log1pf, log1pl — compute a natural logarithm

SYNOPSIS

```
#include <math.h>

double log1p(double x);
float log1pf(float x);
long double log1pl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute $\log_e(1.0 + x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the natural logarithm of $1.0 + x$.

If *x* is −1, a pole error shall occur and *log1p*(), *log1pf*(), and *log1pl*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

For finite values of *x* that are less than −1, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, or +Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *log1p*(), *log1pf*(), and *log1pl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Domain Error

The finite value of *x* is less than −1, or *x* is −Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of *x* is −1.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno*

shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *log*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

log2, log2f, log2l — compute base 2 logarithm functions

SYNOPSIS

```
#include <math.h>

double log2(double x);
float log2f(float x);
long double log2l(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the base 2 logarithm of their argument *x*, $\log_2(x)$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the base 2 logarithm of *x*.

If *x* is ±0, a pole error shall occur and *log2*(), *log2f*(), and *log2l*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

For finite values of *x* that are less than 0, or if *x* is −Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is 1, +0 shall be returned.

If *x* is +Inf, *x* shall be returned.

ERRORS

These functions shall fail if:

Domain Error
: The finite value of *x* is less than zero, or *x* is −Inf.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error
: The value of *x* is zero.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *log*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

logb, logbf, logbl — radix-independent exponent

SYNOPSIS

```
#include <math.h>

double logb(double x);
float logbf(float x);
long double logbl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the exponent of *x*, which is the integral part of $\log_r |x|$, as a signed floating-point value, for non-zero *x*, where *r* is the radix of the machine's floating-point arithmetic, which is the value of FLT_RADIX defined in the <*float.h*> header.

If *x* is subnormal it is treated as though it were normalized; thus for finite positive *x*:

$$1 <= x * \text{FLT\_RADIX}^{-\text{logb}(x)} < \text{FLT\_RADIX}$$

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the exponent of *x*.

If *x* is ±0, *logb*(), *logbf*(), and *logbl*() shall return −HUGE_VAL, −HUGE_VALF, and −HUGE_VALL, respectively.

On systems that support the IEC 60559 Floating-Point option, a pole error shall occur;
otherwise, a pole error may occur.

If *x* is NaN, a NaN shall be returned.

If *x* is ±Inf, +Inf shall be returned.

ERRORS

These functions shall fail if:

Pole Error The value of *x* is ±0.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

These functions may fail if:

Pole Error The value of *x* is 0.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *ilogb*( ), *scalbln*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<float.h>**, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

logf, logl — natural logarithm function

SYNOPSIS

```
#include <math.h>

float logf(float x);
long double logl(long double x);
```

DESCRIPTION

Refer to *log*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

longjmp — non-local goto

SYNOPSIS

#include <setjmp.h>

void longjmp(jmp_buf *env*, int *val*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *longjmp*() function shall restore the environment saved by the most recent invocation of *setjmp*() in the same process, with the corresponding **jmp_buf** argument. If the most recent invocation of *setjmp*() with the corresponding **jmp_buf** occurred in another thread, or if there is no such invocation, or if the function containing the invocation of *setjmp*() has terminated execution in the interim, or if the invocation of *setjmp*() was within the scope of an identifier with variably modified type and execution has left that scope in the interim, the behavior is undefined. It is unspecified whether *longjmp*() restores the signal mask, leaves the signal mask unchanged, or restores it to its value at the time *setjmp*() was called.

All accessible objects have values, and all other components of the abstract machine have state (for example, floating-point status flags and open files), as of the time *longjmp*() was called, except that the values of objects of automatic storage duration are unspecified if they meet all the following conditions:

* They are local to the function containing the corresponding *setjmp*() invocation.
* They do not have volatile-qualified type.
* They are changed between the *setjmp*() invocation and *longjmp*() call.

Although *longjmp*() is an async-signal-safe function, if it is invoked from a signal handler which interrupted a non-async-signal-safe function or equivalent (such as the processing equivalent to *exit*() performed after a return from the initial call to *main*()), the behavior of any subsequent call to a non-async-signal-safe function or equivalent is undefined.

The effect of a call to *longjmp*() where initialization of the **jmp_buf** structure was not performed in the calling thread is undefined.

RETURN VALUE

After *longjmp*() is completed, program execution continues as if the corresponding invocation of *setjmp*() had just returned the value specified by *val*. The *longjmp*() function shall not cause *setjmp*() to return 0; if *val* is 0, *setjmp*() shall return 1.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications whose behavior depends on the value of the signal mask should not use *longjmp*() and *setjmp*(), since their effect on the signal mask is unspecified, but should instead use the *siglongjmp*() and *sigsetjmp*() functions (which can save and restore the signal mask under application control).

It is recommended that applications do not call *longjmp*() or *siglongjmp*() from signal handlers. To avoid undefined behavior when calling these functions from a signal handler, the application needs to ensure one of the following two things:

1. After the call to *longjmp*() or *siglongjmp*() the process only calls async-signal-safe functions and does not return from the initial call to *main*().
2. Any signal whose handler calls *longjmp*() or *siglongjmp*() is blocked during *every* call to a non-async-signal-safe function, and no such calls are made after returning from the initial call to *main*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setjmp( ), *sigaction*( ), *siglongjmp*( ), *sigsetjmp*( )

The Base Definitions volume of POSIX.1-2017, **<setjmp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

```
#include <stdlib.h>

long lrand48(void);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lrint, lrintf, lrintl — round to nearest integer value using current rounding direction

SYNOPSIS

```
#include <math.h>

long lrint(double x);
long lrintf(float x);
long lrintl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding according to the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is −Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur; otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error
: The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error
: The correct value is not representable as an integer.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions provide floating-to-integer conversions. They round according to the current rounding direction. If the rounded value is outside the range of the return type, the numeric result is unspecified and the invalid floating-point exception is raised. When they raise no other floating-point exception and the result differs from the argument, they raise the inexact floating-point exception.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *llrint*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lround, lroundf, lroundl — round to nearest integer value

SYNOPSIS

#include <math.h>

long lround(double *x*);
long lroundf(float *x*);
long lroundl(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value, rounding halfway cases away from zero, regardless of the current rounding direction.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value.

If *x* is NaN, a domain error shall occur and an unspecified value is returned.

If *x* is +Inf, a domain error shall occur and an unspecified value is returned.

If *x* is −Inf, a domain error shall occur and an unspecified value is returned.

If the correct value is positive and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur; otherwise, a domain error may occur.

If the correct value is negative and too large to represent as a **long**, an unspecified value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain shall occur; otherwise, a domain error may occur.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is NaN or ±Inf, or the correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error

The correct value is not representable as an integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions differ from the *lrint*() functions in the default rounding direction, with the *lround*() functions rounding halfway cases away from zero and needing not to raise the inexact floating-point exception for non-integer arguments that round to within the range of the return type.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *llround*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lsearch, lfind — linear search and update

SYNOPSIS

```
#include <search.h>

void *lsearch(const void *key, void *base, size_t *nelp, size_t width,
   int (*compar)(const void *, const void *));
void *lfind(const void *key, const void *base, size_t *nelp,
   size_t width, int (*compar)(const void *, const void *));
```

DESCRIPTION

The *lsearch*() function shall linearly search the table and return a pointer into the table for the matching entry. If the entry does not occur, it shall be added at the end of the table. The *key* argument points to the entry to be sought in the table. The *base* argument points to the first element in the table. The *width* argument is the size of an element in bytes. The *nelp* argument points to an integer containing the current number of elements in the table. The integer to which *nelp* points shall be incremented if the entry is added to the table. The *compar* argument points to a comparison function which the application shall supply (for example, *strcmp*()). It is called with two arguments that point to the elements being compared. The application shall ensure that the function returns 0 if the elements are equal, and non-zero otherwise.

The *lfind*() function shall be equivalent to *lsearch*(), except that if the entry is not found, it is not added to the table. Instead, a null pointer is returned.

RETURN VALUE

If the searched for entry is found, both *lsearch*() and *lfind*() shall return a pointer to it. Otherwise, *lfind*() shall return a null pointer and *lsearch*() shall return a pointer to the newly added element.

Both functions shall return a null pointer in case of error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Storing Strings in a Table

This fragment reads in less than or equal to TABSIZE strings of length less than or equal to ELSIZE and stores them in a table, eliminating duplicates.

```
#include <stdio.h>
#include <string.h>
#include <search.h>

#define TABSIZE 50
#define ELSIZE 120

...
   char line[ELSIZE], tab[TABSIZE][ELSIZE];
   size_t nel = 0;
   ...
   while (fgets(line, ELSIZE, stdin) != NULL && nel < TABSIZE)
      (void) lsearch(line, tab, &nel,
         ELSIZE, (int (*)(const void *, const void *)) strcmp);
   ...
```

Finding a Matching Entry

The following example finds any line that reads **"This**is**a**test.".

```
#include <search.h>
#include <string.h>
...
char line[ELSIZE], tab[TABSIZE][ELSIZE];
size_t nel = 0;
char *findline;
void *entry;

findline = "This is a test.\n";

entry = lfind(findline, tab, &nel, ELSIZE, (
    int (*)(const void *, const void *)) strcmp);
```

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

Undefined results can occur if there is not enough room in the table to add a new item.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *tdelete*( )

The Base Definitions volume of POSIX.1-2017, **<search.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lseek — move the read/write file offset

SYNOPSIS

```
#include <unistd.h>

off_t lseek(int fildes, off_t offset, int whence);
```

DESCRIPTION

The *lseek*() function shall set the file offset for the open file description associated with the file descriptor *fildes,* as follows:

* If *whence* is SEEK_SET, the file offset shall be set to *offset* bytes.

* If *whence* is SEEK_CUR, the file offset shall be set to its current location plus *offset*.

* If *whence* is SEEK_END, the file offset shall be set to the size of the file plus *offset*.

The symbolic constants SEEK_SET, SEEK_CUR, and SEEK_END are defined in *<unistd.h>*.

The behavior of *lseek*() on devices which are incapable of seeking is implementation-defined. The value of the file offset associated with such a device is undefined.

The *lseek*() function shall allow the file offset to be set beyond the end of the existing data in the file. If data is later written at this point, subsequent reads of data in the gap shall return bytes with the value 0 until data is actually written into the gap.

The *lseek*() function shall not, by itself, extend the size of a file.

If *fildes* refers to a shared memory object, the result of the *lseek*() function is unspecified.

If *fildes* refers to a typed memory object, the result of the *lseek*() function is unspecified.

RETURN VALUE

Upon successful completion, the resulting offset, as measured in bytes from the beginning of the file, shall be returned. Otherwise, −1 shall be returned, *errno* shall be set to indicate the error, and the file offset shall remain unchanged.

ERRORS

The *lseek*() function shall fail if:

EBADF
The *fildes* argument is not an open file descriptor.

EINVAL
The *whence* argument is not a proper value, or the resulting file offset would be negative for a regular file, block special file, or directory.

EOVERFLOW
The resulting file offset would be a value which cannot be represented correctly in an object of type **off_t**.

ESPIPE
The *fildes* argument is associated with a pipe, FIFO, or socket.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The ISO C standard includes the functions *fgetpos*() and *fsetpos*(), which work on very large files by use of a special positioning type.

Although *lseek*() may position the file offset beyond the end of the file, this function does not itself extend the size of the file. While the only function in POSIX.1-2008 that may directly extend the size of the file is *write*(), *truncate*(), and *ftruncate*(), several functions originally derived from the ISO C standard, such as *fwrite*(), *fprintf*(), and so on, may do so (by causing calls on *write*()).

An invalid file offset that would cause **[EINVAL]** to be returned may be both implementation-defined and device-dependent (for example, memory may have few invalid values). A negative file offset may be valid for some devices in some implementations.

The POSIX.1-1990 standard did not specifically prohibit *lseek*() from returning a negative offset. Therefore, an application was required to clear *errno* prior to the call and check *errno* upon return to determine whether a return value of (**off_t**)−1 is a negative offset or an indication of an error condition. The standard developers did not wish to require this action on the part of a conforming application, and chose to require that *errno* be set to **[EINVAL]** when the resulting file offset would be negative for a regular file, block special file, or directory.

FUTURE DIRECTIONS

None.

SEE ALSO

open( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

lstat — get file status

SYNOPSIS

```
#include <sys/stat.h>

int lstat(const char *restrict path, struct stat *restrict buf);
```

DESCRIPTION

Refer to *fstatat*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

malloc — a memory allocator

SYNOPSIS

```
#include <stdlib.h>

void *malloc(size_t size);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *malloc*() function shall allocate unused space for an object whose size in bytes is specified by *size* and whose value is unspecified.

The order and contiguity of storage allocated by successive calls to *malloc*() is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned points to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned. If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the returned pointer is used to access an object.

RETURN VALUE

Upon successful completion with *size* not equal to 0, *malloc*() shall return a pointer to the allocated space. If *size* is 0, either:

* A null pointer shall be returned and *errno* may be set to an implementation-defined value, or
* A pointer to the allocated space shall be returned. The application shall ensure that the pointer is not used to access an object.

Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *malloc*() function shall fail if:

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

calloc( ), *free*( ), *getrlimit*( ), *posix_memalign*( ), *realloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mblen — get number of bytes in a character

SYNOPSIS

```
#include <stdlib.h>

int mblen(const char *s, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *s* is not a null pointer, *mblen*() shall determine the number of bytes constituting the character pointed to by *s*. Except that the shift state of *mbtowc*() is not affected, it shall be equivalent to:

```
    mbtowc((wchar_t *)0, s, n);
```

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *mblen*().

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

The *mblen*() function need not be thread-safe.

RETURN VALUE

If *s* is a null pointer, *mblen*() shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mblen*() shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the character (if the next *n* or fewer bytes form a valid character), or return −1 (if they do not form a valid character) and may set *errno* to indicate the error. In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mblen*() function may fail if:

EILSEQ
An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbtowc( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbrlen — get number of bytes in a character (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrlen(const char *restrict s, size_t n,
   mbstate_t *restrict ps);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *s* is not a null pointer, *mbrlen*() shall determine the number of bytes constituting the character pointed to by *s*. It shall be equivalent to:

```
mbstate_t internal;
mbrtowc(NULL, s, n, ps != NULL ? ps : &internal);
```

If *ps* is a null pointer, the *mbrlen*() function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *mbrlen*().

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

The *mbrlen*() function need not be thread-safe if called with a NULL *ps* argument.

The *mbrlen*() function shall not change the setting of *errno* if successful.

RETURN VALUE

The *mbrlen*() function shall return the first of the following that applies:

0	If the next *n* or fewer bytes complete the character that corresponds to the null wide character.
positive	If the next *n* or fewer bytes complete a valid character; the value returned shall be the number of bytes that complete the character.
(**size_t**)−2	If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed. When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).
(**size_t**)−1	If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character. In this case, **[EILSEQ]** shall be stored in *errno* and the conversion state is undefined.

ERRORS

The *mbrlen*() function shall fail if:

EILSEQ

An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

The *mbrlen*() function may fail if:

EINVAL
ps points to an object that contains an invalid conversion state.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *mbrtowc*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbrtowc — convert a character to a wide-character code (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbrtowc(wchar_t *restrict pwc, const char *restrict s,
    size_t n, mbstate_t *restrict ps);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *s* is a null pointer, the *mbrtowc*() function shall be equivalent to the call:

```
mbrtowc(NULL, "", 1, ps)
```

In this case, the values of the arguments *pwc* and *n* are ignored.

If *s* is not a null pointer, the *mbrtowc*() function shall inspect at most *n* bytes beginning at the byte pointed to by *s* to determine the number of bytes needed to complete the next character (including any shift sequences). If the function determines that the next character is completed, it shall determine the value of the corresponding wide character and then, if *pwc* is not a null pointer, shall store that value in the object pointed to by *pwc*. If the corresponding wide character is the null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbrtowc*() function shall use its own internal **mbstate_t** object, which shall be initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *mbrtowc*().

The behavior of this function is affected by the *LC_CTYPE* category of the current locale.

The *mbrtowc*() function need not be thread-safe if called with a NULL *ps* argument.

The *mbrtowc*() function shall not change the setting of *errno* if successful.

RETURN VALUE

The *mbrtowc*() function shall return the first of the following that applies:

0
: If the next *n* or fewer bytes complete the character that corresponds to the null wide character (which is the value stored).

between 1 and *n* inclusive
: If the next *n* or fewer bytes complete a valid character (which is the value stored); the value returned shall be the number of bytes that complete the character.

(**size_t**)−2
: If the next *n* bytes contribute to an incomplete but potentially valid character, and all *n* bytes have been processed (no value is stored). When *n* has at least the value of the {MB_CUR_MAX} macro, this case can only occur if *s* points at a sequence of redundant shift sequences (for implementations with state-dependent encodings).

(**size_t**)−1
: If an encoding error occurs, in which case the next *n* or fewer bytes do not contribute to a complete and valid character (no value is stored). In this case, **[EILSEQ]** shall be stored in *errno* and the conversion state is undefined.

ERRORS

The *mbrtowc*() function shall fail if:

EILSEQ

An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

The *mbrtowc*() function may fail if:

EINVAL

ps points to an object that contains an invalid conversion state.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *mbsrtowcs*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbsinit — determine conversion object status

SYNOPSIS

```
#include <wchar.h>

int mbsinit(const mbstate_t *ps);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *ps* is not a null pointer, the *mbsinit*() function shall determine whether the object pointed to by *ps* describes an initial conversion state.

RETURN VALUE

The *mbsinit*() function shall return non-zero if *ps* is a null pointer, or if the pointed-to object describes an initial conversion state; otherwise, it shall return zero.

If an **mbstate_t** object is altered by any of the functions described as "restartable", and is then used with a different character sequence, or in the other conversion direction, or with a different *LC_CTYPE* category setting than on earlier function calls, the behavior is undefined.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The **mbstate_t** object is used to describe the current conversion state from a particular character sequence to a wide-character sequence (or *vice versa*) under the rules of a particular setting of the *LC_CTYPE* category of the current locale.

The initial conversion state corresponds, for a conversion in either direction, to the beginning of a new character sequence in the initial shift state. A zero valued **mbstate_t** object is at least one way to describe an initial conversion state. A zero valued **mbstate_t** object can be used to initiate conversion involving any character sequence, in any *LC_CTYPE* category setting.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbrlen( ), *mbrtowc*( ), *mbsrtowcs*( ), *wcrtomb*( ), *wcsrtombs*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbsnrtowcs, mbsrtowcs — convert a character string to a wide-character string (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t mbsnrtowcs(wchar_t *restrict dst, const char **restrict src,
    size_t nmc, size_t len, mbstate_t *restrict ps);
size_t mbsrtowcs(wchar_t *restrict dst, const char **restrict src,
    size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

For *mbsrtowcs*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *mbsrtowcs*() function shall convert a sequence of characters, beginning in the conversion state described by the object pointed to by *ps*, from the array indirectly pointed to by *src* into a sequence of corresponding wide characters. If *dst* is not a null pointer, the converted characters shall be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null character, which shall also be stored. Conversion shall stop early in either of the following cases:

* A sequence of bytes is encountered that does not form a valid character.
* *len* codes have been stored into the array pointed to by *dst* (and *dst* is not a null pointer).

Each conversion shall take place as if by a call to the *mbrtowc*() function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null character) or the address just past the last character converted (if any). If conversion stopped due to reaching a terminating null character, and if *dst* is not a null pointer, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *mbsrtowcs*() function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

The *mbsnrtowcs*() function shall be equivalent to the *mbsrtowcs*() function, except that the conversion of characters indirectly pointed to by *src* is limited to at most *nmc* bytes (the size of the input buffer), and under conditions where *mbsrtowcs*() would assign the address just past the last character converted (if any) to the pointer object pointed to by *src*, *mbsnrtowcs*() shall instead assign the address just past the last byte processed (if any) to that pointer object. If the input buffer ends with an incomplete character, it is unspecified whether conversion stops at the end of the previous character (if any), or at the end of the input buffer. In the latter case, a subsequent call to *mbsnrtowcs*() with an input buffer that starts with the remainder of the incomplete character shall correctly complete the conversion of that character.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls these functions.

The *mbsnrtowcs*() and *mbsrtowcs*() functions need not be thread-safe if called with a NULL *ps* argument.

The *mbsrtowcs*() function shall not change the setting of *errno* if successful.

RETURN VALUE

If the input conversion encounters a sequence of bytes that do not form a valid character, an encoding error occurs. In this case, these functions shall store the value of the macro **[EILSEQ]** in *errno* and shall return (**size_t**)−1; the conversion state is undefined. Otherwise, these functions shall return the number of characters successfully converted, not including the terminating null (if any).

ERRORS

These functions shall fail if:

EILSEQ
An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

These functions may fail if:

EINVAL
ps points to an object that contains an invalid conversion state.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version may require that when the input buffer ends with an incomplete character, conversion stops at the end of the input buffer.

SEE ALSO

iconv( ), *mbrtowc*( ), *mbsinit*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbstowcs — convert a character string to a wide-character string

SYNOPSIS

```
#include <stdlib.h>

size_t mbstowcs(wchar_t *restrict pwcs, const char *restrict s,
    size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *mbstowcs*() function shall convert a sequence of characters that begins in the initial shift state from the array pointed to by *s* into a sequence of corresponding wide-character codes and shall store not more than *n* wide-character codes into the array pointed to by *pwcs*. No characters that follow a null byte (which is converted into a wide-character code with value 0) shall be examined or converted. Each character shall be converted as if by a call to *mbtowc*(), except that the shift state of *mbtowc*() is not affected.

No more than *n* elements shall be modified in the array pointed to by *pwcs*. If copying takes place between objects that overlap, the behavior is undefined.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale. If *pwcs* is a null pointer, *mbstowcs*() shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

RETURN VALUE

If an invalid character is encountered, *mbstowcs*() shall return (**size_t**)−1 and shall set *errno* to indicate the error.

Otherwise, *mbstowcs*() shall return the number of the array elements modified (or required if *pwcs* is null), not including a terminating 0 code, if any. The array shall not be zero-terminated if the value returned is *n*.

ERRORS

The *mbstowcs*() function shall fail if:

EILSEQ

An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *wctomb*( ), *wcstombs*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mbtowc — convert a character to a wide-character code

SYNOPSIS

```
#include <stdlib.h>

int mbtowc(wchar_t *restrict pwc, const char *restrict s, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *s* is not a null pointer, *mbtowc*() shall determine the number of bytes that constitute the character pointed to by *s*. It shall then determine the wide-character code for the value of type **wchar_t** that corresponds to that character. (The value of the wide-character code corresponding to the null byte is 0.) If the character is valid and *pwc* is not a null pointer, *mbtowc*() shall store the wide-character code in the object pointed to by *pwc*.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function is placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. If the implementation employs special bytes to change the shift state, these bytes shall not produce separate wide-character codes, but shall be grouped with an adjacent character. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified. At most *n* bytes of the array pointed to by *s* shall be examined.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *mbtowc*().

The *mbtowc*() function need not be thread-safe.

RETURN VALUE

If *s* is a null pointer, *mbtowc*() shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *mbtowc*() shall either return 0 (if *s* points to the null byte), or return the number of bytes that constitute the converted character (if the next *n* or fewer bytes form a valid character), or return −1 and shall set *errno* to indicate the error (if they do not form a valid character).

In no case shall the value returned be greater than *n* or the value of the {MB_CUR_MAX} macro.

ERRORS

The *mbtowc*() function shall fail if:

EILSEQ

An invalid character sequence is detected. In the POSIX locale an **[EILSEQ]** error cannot occur since all byte values are valid characters.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbstowcs*( ), *wctomb*( ), *wcstombs*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memccpy — copy bytes in memory

SYNOPSIS

```
#include <string.h>

void *memccpy(void *restrict s1, const void *restrict s2,
   int c, size_t n);
```

DESCRIPTION

The *memccpy*() function shall copy bytes from memory area *s2* into *s1*, stopping after the first occurrence of byte *c* (converted to an **unsigned char**) is copied, or after *n* bytes are copied, whichever comes first. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *memccpy*() function shall return a pointer to the byte after the copy of *c* in *s1*, or a null pointer if *c* was not found in the first *n* bytes of *s2*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *memccpy*() function does not check for the overflow of the receiving memory area.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memchr — find byte in memory

SYNOPSIS

#include <string.h>

void *memchr(const void **s*, int *c*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *memchr*() function shall locate the first occurrence of *c* (converted to an **unsigned char**) in the initial *n* bytes (each interpreted as **unsigned char**) pointed to by *s*.

Implementations shall behave as if they read the memory byte by byte from the beginning of the bytes pointed to by *s* and stop at the first occurrence of *c* (if it is found in the initial *n* bytes).

RETURN VALUE

The *memchr*() function shall return a pointer to the located byte, or a null pointer if the byte is not found.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memcmp — compare bytes in memory

SYNOPSIS

#include <string.h>

int memcmp(const void **s1*, const void **s2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *memcmp*() function shall compare the first *n* bytes (each interpreted as **unsigned char**) of the object pointed to by *s1* to the first *n* bytes of the object pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the objects being compared.

RETURN VALUE

The *memcmp*() function shall return an integer greater than, equal to, or less than 0, if the object pointed to by *s1* is greater than, equal to, or less than the object pointed to by *s2*, respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memcpy — copy bytes in memory

SYNOPSIS

```
#include <string.h>

void *memcpy(void *restrict s1, const void *restrict s2, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *memcpy*() function shall copy *n* bytes from the object pointed to by *s2* into the object pointed to by *s1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *memcpy*() function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *memcpy*() function does not check for the overflow of the receiving memory area.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memmove — copy bytes in memory with overlapping areas

SYNOPSIS

#include <string.h>

void *memmove(void **s1*, const void **s2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *memmove*() function shall copy *n* bytes from the object pointed to by *s2* into the object pointed to by *s1*. Copying takes place as if the *n* bytes from the object pointed to by *s2* are first copied into a temporary array of *n* bytes that does not overlap the objects pointed to by *s1* and *s2*, and then the *n* bytes from the temporary array are copied into the object pointed to by *s1*.

RETURN VALUE

The *memmove*() function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

memset — set bytes in memory

SYNOPSIS

#include <string.h>

void *memset(void **s*, int *c*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *memset*() function shall copy *c* (converted to an **unsigned char**) into each of the first *n* bytes of the object pointed to by *s*.

RETURN VALUE

The *memset*() function shall return *s*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mkdir, mkdirat — make a directory

SYNOPSIS

#include <sys/stat.h>

int mkdir(const char **path*, mode_t *mode*);

#include <fcntl.h>

int mkdirat(int *fd*, const char **path*, mode_t *mode*);

DESCRIPTION

The *mkdir*() function shall create a new directory with name *path*. The file permission bits of the new directory shall be initialized from *mode*. These file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the meaning of these additional bits is implementation-defined.

The directory's user ID shall be set to the process' effective user ID. The directory's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the directory's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the directory's group ID to the effective group ID of the calling process.

The newly created directory shall be an empty directory.

If *path* names a symbolic link, *mkdir*() shall fail and set *errno* to **[EEXIST]**.

Upon successful completion, *mkdir*() shall mark for update the last data access, last data modification, and last file status change timestamps of the directory. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *mkdirat*() function shall be equivalent to the *mkdir*() function except in the case where *path* specifies a relative path. In this case the newly created directory is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *mkdirat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *mkdir*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, no directory shall be created.

ERRORS

These functions shall fail if:

EACCES
Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.

EEXIST
The named file exists.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

EMLINK
The link count of the parent directory would exceed {LINK_MAX}.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the path prefix specified by *path* does not name an existing directory or *path* is an empty string.

ENOSPC
The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
The parent directory resides on a read-only file system.

In addition, the *mkdirat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Creating a Directory

The following example shows how to create a directory named **/home/cnd/mod1**, with read/write/search permissions for owner and group, and with read/search permissions for others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkdir("/home/cnd/mod1", S_IRWXU | S_IRWXG | S_IROTH | S_IXOTH);
```

APPLICATION USAGE

None.

RATIONALE

The *mkdir*() function originated in 4.2 BSD and was added to System V in Release 3.0.

4.3 BSD detects **[ENAMETOOLONG]**.

The POSIX.1-1990 standard required that the group ID of a newly created directory be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*() to set the group ID after the directory is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mkdirat*() function is to create a directory in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to the call to *mkdir*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkdirat*() function it can be guaranteed that the newly created directory is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *mkdtemp*( ), *mknod*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mkdtemp, mkstemp — create a unique directory or file

SYNOPSIS

```
#include <stdlib.h>

char *mkdtemp(char *template);
int mkstemp(char *template);
```

DESCRIPTION

The *mkdtemp*() function shall create a directory with a unique name derived from *template*. The application shall ensure that the string provided in *template* is a pathname ending with at least six trailing **'X'** characters. The *mkdtemp*() function shall modify the contents of *template* by replacing six or more **'X'** characters at the end of the pathname with the same number of characters from the portable filename character set. The characters shall be chosen such that the resulting pathname does not duplicate the name of an existing file at the time of the call to *mkdtemp*(). The *mkdtemp*() function shall use the resulting pathname to create the new directory as if by a call to:

```
mkdir(pathname, S_IRWXU)
```

The *mkstemp*() function shall create a regular file with a unique name derived from *template* and return a file descriptor for the file open for reading and writing. The application shall ensure that the string provided in *template* is a pathname ending with at least six trailing **'X'** characters. The *mkstemp*() function shall modify the contents of *template* by replacing six or more **'X'** characters at the end of the pathname with the same number of characters from the portable filename character set. The characters shall be chosen such that the resulting pathname does not duplicate the name of an existing file at the time of the call to *mkstemp*(). The *mkstemp*() function shall use the resulting pathname to create the file, and obtain a file descriptor for it, as if by a call to:

```
open(pathname, O_RDWR|O_CREAT|O_EXCL, S_IRUSR|S_IWUSR)
```

By behaving as if the O_EXCL flag for *open*() is set, the function prevents any possible race condition between testing whether the file exists and opening it for use.

RETURN VALUE

Upon successful completion, the *mkdtemp*() function shall return the value of *template*. Otherwise, it shall return a null pointer and shall set *errno* to indicate the error.

Upon successful completion, the *mkstemp*() function shall return an open file descriptor. Otherwise, it shall return −1 and shall set *errno* to indicate the error.

ERRORS

The *mkdtemp*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be created.

EINVAL
The string pointed to by *template* does not end in **"XXXXXX"**.

ELOOP
A loop exists in symbolic links encountered during resolution of the path of the directory to be created.

EMLINK
The link count of the parent directory would exceed {LINK_MAX}.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the path prefix specified by the *template* argument does not name an existing directory.

ENOSPC
The file system does not contain enough space to hold the contents of the new directory or to extend the parent directory of the new directory.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
The parent directory resides on a read-only file system.

The *mkdtemp*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the path of the directory to be created.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The error conditions for the *mkstemp*() function are defined in *open*( ).

The following sections are informative.

EXAMPLES

Generating a Pathname

The following example creates a file with a 10-character name beginning with the characters **"file"** and opens the file for reading and writing. The value returned as the value of *fd* is a file descriptor that identifies the file.

```
#include <stdlib.h>
...
char template[] = "/tmp/fileXXXXXX";
int fd;

fd = mkstemp(template);
```

APPLICATION USAGE

It is possible to run out of letters.

Portable applications should pass exactly six trailing **'X'**s in the template and no more; implementations may treat any additional trailing **'X'**s as either a fixed or replaceable part of the template. To be sure of only passing six, a fixed string of at least one non-**'X'** character should precede the six **'X'**s.

Since **'X'** is in the portable filename character set, some of the replacement characters can be **'X'**s, leaving part (or even all) of the template effectively unchanged.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getpid( ), *mkdir*( ), *open*( ), *tmpfile*( ), *tmpnam*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mkfifo, mkfifoat — make a FIFO special file

SYNOPSIS

#include <sys/stat.h>

int mkfifo(const char **path*, mode_t *mode*);

#include <fcntl.h>

int mkfifoat(int *fd*, const char **path*, mode_t *mode*);

DESCRIPTION

The *mkfifo*() function shall create a new FIFO special file named by the pathname pointed to by *path*. The file permission bits of the new FIFO shall be initialized from *mode*. The file permission bits of the *mode* argument shall be modified by the process' file creation mask.

When bits in *mode* other than the file permission bits are set, the effect is implementation-defined.

If *path* names a symbolic link, *mkfifo*() shall fail and set *errno* to **[EEXIST]**.

The FIFO's user ID shall be set to the process' effective user ID. The FIFO's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the FIFO's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the FIFO's group ID to the effective group ID of the calling process.

Upon successful completion, *mkfifo*() shall mark for update the last data access, last data modification, and last file status change timestamps of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *mkfifoat*() function shall be equivalent to the *mkfifo*() function except in the case where *path* specifies a relative path. In this case the newly created FIFO is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *mkfifoat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *mkfifo*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, no FIFO shall be created.

ERRORS

These functions shall fail if:

EACCES
A component of the path prefix denies search permission, or write permission is denied on the parent directory of the FIFO to be created.

EEXIST
The named file already exists.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the path prefix of *path* does not name an existing file or *path* is an empty string.

ENOENT or **ENOTDIR**
The *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory that would contain the new file cannot be extended or the file system is out of file-allocation resources.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
The named file resides on a read-only file system.

The *mkfifoat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Creating a FIFO File

The following example shows how to create a FIFO file named **/home/cnd/mod_done**, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

int status;
...
status = mkfifo("/home/cnd/mod_done", S_IWUSR | S_IRUSR |
   S_IRGRP | S_IROTH);
```

APPLICATION USAGE

None.

RATIONALE

The syntax of this function is intended to maintain compatibility with historical implementations of *mknod*(). The latter function was included in the 1984 /usr/group standard but only for use in creating FIFO special files. The *mknod*() function was originally excluded from the POSIX.1-1988 standard as implementation-defined and replaced by *mkdir*() and *mkfifo*(). The *mknod*() function is now included for alignment with the Single UNIX Specification.

The POSIX.1-1990 standard required that the group ID of a newly created FIFO be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*() to set the group ID after the FIFO is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mkfifoat*() function is to create a FIFO special file in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mkfifo*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mkfifoat*() function it can be guaranteed that the newly created FIFO is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *mknod*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mknod, mknodat — make directory, special file, or regular file

SYNOPSIS

#include <sys/stat.h>

int mknod(const char **path*, mode_t *mode*, dev_t *dev*);

#include <fcntl.h>

int mknodat(int *fd*, const char **path*, mode_t *mode*, dev_t *dev*);

DESCRIPTION

The *mknod*() function shall create a new file named by the pathname to which the argument *path* points.

The file type for *path* is OR'ed into the *mode* argument, and the application shall select one of the following symbolic constants:

tab(!) box center; cB | cB lw(1i) | lw(3i). Name!Description _ S_IFIFO!FIFO-special S_IFCHR!Character-special (non-portable) S_IFDIR!Directory (non-portable) S_IFBLK!Block-special (non-portable) S_IFREG!Regular (non-portable)

The only portable use of *mknod*() is to create a FIFO-special file. If *mode* is not S_IFIFO or *dev* is not 0, the behavior of *mknod*() is unspecified.

The permissions for the new file are OR'ed into the *mode* argument, and may be selected from any combination of the following symbolic constants:

tab(!) box center; cB | cB lw(1i) | lw(3i). Name!Description _ S_ISUID!Set user ID on execution. S_ISGID!Set group ID on execution. S_IRWXU!Read, write, or execute (search) by owner. S_IRUSR!Read by owner. S_IWUSR!Write by owner. S_IXUSR!Execute (search) by owner. S_IRWXG!Read, write, or execute (search) by group. S_IRGRP!Read by group. S_IWGRP!Write by group. S_IXGRP!Execute (search) by group. S_IRWXO!Read, write, or execute (search) by others. S_IROTH!Read by others. S_IWOTH!Write by others. S_IXOTH!Execute (search) by others. S_ISVTX!On directories, restricted deletion flag.

The user ID of the file shall be initialized to the effective user ID of the process. The group ID of the file shall be initialized to either the effective group ID of the process or the group ID of the parent directory. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process. The owner, group, and other permission bits of *mode* shall be modified by the file mode creation mask of the process. The *mknod*() function shall clear each bit whose corresponding bit in the file mode creation mask of the process is set.

If *path* names a symbolic link, *mknod*() shall fail and set *errno* to **[EEXIST]**.

Upon successful completion, *mknod*() shall mark for update the last data access, last data modification, and last file status change timestamps of the file. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

Only a process with appropriate privileges may invoke *mknod*() for file types other than FIFO-special.

The *mknodat*() function shall be equivalent to the *mknod*() function except in the case where *path* specifies a relative path. In this case the newly created directory, special file, or regular file is located relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *mknodat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *mknod*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, the new file shall not be created.

ERRORS

These functions shall fail if:

EACCES
A component of the path prefix denies search permission, or write permission is denied on the parent directory.

EEXIST
The named file exists.

EINVAL
An invalid argument exists.

EIO An I/O error occurred while accessing the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the path prefix of *path* does not name an existing file or *path* is an empty string.

ENOENT or **ENOTDIR**
The *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory that would contain the new file cannot be extended or the file system is out of file allocation resources.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The invoking process does not have appropriate privileges and the file type is not FIFO-special.

EROFS
The directory in which the file is to be created is located on a read-only file system.

The *mknodat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Creating a FIFO Special File

The following example shows how to create a FIFO special file named **/home/cnd/mod_done**, with read/write permissions for owner, and with read permissions for group and others.

```
#include <sys/types.h>
#include <sys/stat.h>

dev_t dev;
int   status;
...
status  = mknod("/home/cnd/mod_done", S_IFIFO | S_IWUSR |
    S_IRUSR | S_IRGRP | S_IROTH, dev);
```

APPLICATION USAGE

The *mkfifo*() function is preferred over this function for making FIFO special files.

RATIONALE

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*() to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *mknodat*() function is to create directories, special files, or regular files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *mknod*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *mknodat*() function it can be guaranteed that the newly created directory, special file, or regular file is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *exec*, *fstatat*( ), *mkdir*( ), *mkfifo*( ), *open*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_stat.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mkstemp — create a unique file

SYNOPSIS

```
#include <stdlib.h>

int mkstemp(char *template);
```

DESCRIPTION

Refer to *mkdtemp*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mktime — convert broken-down time into time since the Epoch

SYNOPSIS

```
#include <time.h>

time_t mktime(struct tm *timeptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *mktime*() function shall convert the broken-down time, expressed as local time, in the structure pointed to by *timeptr*, into a time since the Epoch value with the same encoding as that of the values returned by *time*(). The original values of the *tm_wday* and *tm_yday* components of the structure shall be ignored, and the original values of the other components shall not be restricted to the ranges described in *<time.h>*.

A positive or 0 value for *tm_isdst* shall cause *mktime*() to presume initially that Daylight Savings Time, respectively, is or is not in effect for the specified time. A negative value for *tm_isdst* shall cause *mktime*() to attempt to determine whether Daylight Savings Time is in effect for the specified time.

Local timezone information shall be set as though *mktime*() called *tzset*().

The relationship between the **tm** structure (defined in the *<time.h>* header) and the time in seconds since the Epoch is that the result shall be as specified in the expression given in the definition of seconds since the Epoch (see the Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*) corrected for timezone and any seasonal time adjustments, where the names other than *tm_yday* in the structure and in the expression correspond, and the *tm_yday* value used in the expression is the day of the year from 0 to 365 inclusive, calculated from the other **tm** structure members specified in *<time.h>* (excluding *tm_wday*).

Upon successful completion, the values of the *tm_wday* and *tm_yday* components of the structure shall be set appropriately, and the other components shall be set to represent the specified time since the Epoch, but with their values forced to the ranges indicated in the *<time.h>* entry; the final value of *tm_mday* shall not be set until *tm_mon* and *tm_year* are determined.

RETURN VALUE

The *mktime*() function shall return the specified time since the Epoch encoded as a value of type **time_t**. If the time since the Epoch cannot be represented, the function shall return the value (**time_t**)−1 and set *errno* to indicate the error.

ERRORS

The *mktime*() function shall fail if:

EOVERFLOW
The result cannot be represented.

The following sections are informative.

EXAMPLES

What day of the week is July 4, 2001?

```
#include <stdio.h>
#include <time.h>

struct tm time_str;
```

```
char daybuf[20];

int main(void)
{
    time_str.tm_year = 2001 — 1900;
    time_str.tm_mon = 7 — 1;
    time_str.tm_mday = 4;
    time_str.tm_hour = 0;
    time_str.tm_min = 0;
    time_str.tm_sec = 1;
    time_str.tm_isdst = -1;
    if (mktime(&time_str) == -1)
        (void)puts("-unknown-");
    else {
        (void)strftime(daybuf, sizeof(daybuf), "%A", &time_str);
        (void)puts(daybuf);
    }
    return 0;
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *gmtime*( ), *localtime*( ), *strftime*( ), *strptime*( ), *time*( ), *tzset*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.16*, *Seconds Since the Epoch*, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mlock, munlock — lock or unlock a range of process address space (**REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int mlock(const void *addr, size_t len);
int munlock(const void *addr, size_t len);
```

DESCRIPTION

The *mlock*() function shall cause those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes to be memory-resident until unlocked or until the process exits or *exec*s another process image. The implementation may require that *addr* be a multiple of {PAGESIZE}.

The *munlock*() function shall unlock those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes, regardless of how many times *mlock*() has been called by the process for any of the pages in the specified range. The implementation may require that *addr* be a multiple of {PAGESIZE}.

If any of the pages in the range specified to a call to *munlock*() are also mapped into the address spaces of other processes, any locks established on those pages by another process are unaffected by the call of this process to *munlock*(). If any of the pages in the range specified by a call to *munlock*() are also mapped into other portions of the address space of the calling process outside the range specified, any locks established on those pages via the other mappings are also unaffected by this call.

Upon successful return from *mlock*(), pages in the specified range shall be locked and memory-resident. Upon successful return from *munlock*(), pages in the specified range shall be unlocked with respect to the address space of the process. Memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlock*().

RETURN VALUE

Upon successful completion, the *mlock*() and *munlock*() functions shall return a value of zero. Otherwise, no change is made to any locks in the address space of the process, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mlock*() and *munlock*() functions shall fail if:

ENOMEM
: Some or all of the address range specified by the *addr* and *len* arguments does not correspond to valid mapped pages in the address space of the process.

The *mlock*() function shall fail if:

EAGAIN
: Some or all of the memory identified by the operation could not be locked when the call was made.

The *mlock*() and *munlock*() functions may fail if:

EINVAL
: The *addr* argument is not a multiple of {PAGESIZE}.

The *mlock*() function may fail if:

ENOMEM
: Locking the pages mapped by the specified range would exceed an implementation-defined limit on the amount of memory that the process may lock.

EPERM
The calling process does not have appropriate privileges to perform the requested operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*( ), *fork*( ), *mlockall*( ), *munmap*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mlockall, munlockall — lock/unlock the address space of a process (**REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int mlockall(int flags);
int munlockall(void);
```

DESCRIPTION

The *mlockall*() function shall cause all of the pages mapped by the address space of a process to be memory-resident until unlocked or until the process exits or *exec*s another process image. The *flags* argument determines whether the pages to be locked are those currently mapped by the address space of the process, those that are mapped in the future, or both. The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following symbolic constants, defined in *<sys/mman.h>*:

MCL_CURRENT
: Lock all of the pages currently mapped into the address space of the process.

MCL_FUTURE
: Lock all of the pages that become mapped into the address space of the process in the future, when those mappings are established.

If MCL_FUTURE is specified, and the automatic locking of future mappings eventually causes the amount of locked memory to exceed the amount of available physical memory or any other implementation-defined limit, the behavior is implementation-defined. The manner in which the implementation informs the application of these situations is also implementation-defined.

The *munlockall*() function shall unlock all currently mapped pages of the address space of the process. Any pages that become mapped into the address space of the process after a call to *munlockall*() shall not be locked, unless there is an intervening call to *mlockall*() specifying MCL_FUTURE or a subsequent call to *mlockall*() specifying MCL_CURRENT. If pages mapped into the address space of the process are also mapped into the address spaces of other processes and are locked by those processes, the locks established by the other processes shall be unaffected by a call by this process to *munlockall*().

Upon successful return from the *mlockall*() function that specifies MCL_CURRENT, all currently mapped pages of the address space of the process shall be memory-resident and locked. Upon return from the *munlockall*() function, all currently mapped pages of the address space of the process shall be unlocked with respect to the address space of the process. The memory residency of unlocked pages is unspecified.

Appropriate privileges are required to lock process memory with *mlockall*().

RETURN VALUE

Upon successful completion, the *mlockall*() function shall return a value of zero. Otherwise, no additional memory shall be locked, and the function shall return a value of −1 and set *errno* to indicate the error. The effect of failure of *mlockall*() on previously existing locks in the address space is unspecified.

If it is supported by the implementation, the *munlockall*() function shall always return a value of zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mlockall*() function shall fail if:

EAGAIN
: Some or all of the memory identified by the operation could not be locked when the call was made.

EINVAL
The *flags* argument is zero, or includes unimplemented flags.

The *mlockall*() function may fail if:

ENOMEM
Locking all of the pages currently mapped into the address space of the process would exceed an implementation-defined limit on the amount of memory that the process may lock.

EPERM
The calling process does not have appropriate privileges to perform the requested operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *exit*( ), *fork*( ), *mlock*( ), *munmap*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mmap — map pages of memory

SYNOPSIS

```
#include <sys/mman.h>

void *mmap(void *addr, size_t len, int prot, int flags,
   int fildes, off_t off);
```

DESCRIPTION

The *mmap*() function shall establish a mapping between an address space of a process and a memory object.

The *mmap*() function shall be supported for the following memory objects:

* Regular files
* Shared memory objects
* Typed memory objects

Support for any other type of file is unspecified.

The format of the call is as follows:

pa=*mmap*(*addr*, *len*, *prot*, *flags*, *fildes*, *off*);

The *mmap*() function shall establish a mapping between the address space of the process at an address *pa* for *len* bytes to the memory object represented by the file descriptor *fildes* at offset *off* for *len* bytes. The value of *pa* is an implementation-defined function of the parameter *addr* and the values of *flags*, further described below. A successful *mmap*() call shall return *pa* as its result. The address range starting at *pa* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) address space of the process. The range of bytes starting at *off* and continuing for *len* bytes shall be legitimate for the possible (not necessarily current) offsets in the memory object represented by *fildes*.

If *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the memory object to be mapped shall be that portion of the typed memory object allocated by the implementation as specified below. In this case, if *off* is non-zero, the behavior of *mmap*() is undefined. If *fildes* refers to a valid typed memory object that is not accessible from the calling process, *mmap*() shall fail.

The mapping established by *mmap*() shall replace any previous mappings for those whole pages containing any part of the address space of the process starting at *pa* and continuing for *len* bytes.

If the size of the mapped file changes after the call to *mmap*() as a result of some other operation on the mapped file, the effect of references to portions of the mapped region that correspond to added or removed portions of the file is unspecified.

If *len* is zero, *mmap*() shall fail and no mapping shall be established.

The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* shall be either PROT_NONE or the bitwise-inclusive OR of one or more of the other flags in the following table, defined in the *<sys/mman.h>* header.

center box tab(!); cB | cB lw(1.5i) | lw(2i). Symbolic Constant!Description _ PROT_READ!Data can be read. PROT_WRITE!Data can be written. PROT_EXEC!Data can be executed. PROT_NONE!Data cannot be accessed.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mmap*()

shall fail.

An implementation may permit accesses other than those specified by *prot*; however, the implementation shall not permit a write to succeed where PROT_WRITE has not been set and shall not permit any access where PROT_NONE alone has been set. The implementation shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. The file descriptor *fildes* shall have been opened with read permission, regardless of the protection options specified. If PROT_WRITE is specified, the application shall ensure that it has opened the file descriptor *fildes* with write permission unless MAP_PRIVATE is specified in the *flags* parameter as described below.

The parameter *flags* provides other information about the handling of the mapped data. The value of *flags* is the bitwise-inclusive OR of these options, defined in <*sys/mman.h*>:

center box tab(!); cB | cB lw(1.5i) | lw(2i). Symbolic Constant!Description _ MAP_SHARED!Changes are shared. MAP_PRIVATE!Changes are private. MAP_FIXED!Interpret *addr* exactly.

It is implementation-defined whether MAP_FIXED shall be supported. MAP_FIXED shall be supported on XSI-conformant systems.

MAP_SHARED and MAP_PRIVATE describe the disposition of write references to the memory object. If MAP_SHARED is specified, write references shall change the underlying object. If MAP_PRIVATE is specified, modifications to the mapped data by the calling process shall be visible only to the calling process and shall not change the underlying object. It is unspecified whether modifications to the underlying object done after the MAP_PRIVATE mapping is established are visible through the MAP_PRIVATE mapping. Either MAP_SHARED or MAP_PRIVATE can be specified, but not both. The mapping type is retained across *fork*().

The state of synchronization objects such as mutexes, semaphores, barriers, and conditional variables placed in shared memory mapped with MAP_SHARED becomes undefined when the last region in any process containing the synchronization object is unmapped.

When *fildes* represents a typed memory object opened with either the POSIX_TYPED_MEM_ALLOCATE flag or the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, *mmap*() shall, if there are enough resources available, map *len* bytes allocated from the corresponding typed memory object which were not previously allocated to any process in any processor that may access that typed memory object. If there are not enough resources available, the function shall fail. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, these allocated bytes shall be contiguous within the typed memory object. If *fildes* represents a typed memory object opened with the POSIX_TYPED_MEM_ALLOCATE flag, these allocated bytes may be composed of non-contiguous fragments within the typed memory object. If *fildes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag nor the POSIX_TYPED_MEM_ALLOCATE flag, *len* bytes starting at offset *off* within the typed memory object are mapped, exactly as when mapping a file or shared memory object. In this case, if two processes map an area of typed memory using the same *off* and *len* values and using file descriptors that refer to the same memory pool (either from the same port or from a different port), both processes shall map the same region of storage.

When MAP_FIXED is set in the *flags* argument, the implementation is informed that the value of *pa* shall be *addr*, exactly. If MAP_FIXED is set, *mmap*() may return MAP_FAILED and set *errno* to **[EINVAL]**. If a MAP_FIXED request is successful, then any previous mappings or memory locks for those whole pages containing any part of the address range [*pa*,*pa*+*len*) shall be removed, as if by an appropriate call to *munmap*(), before the new mapping is established.

When MAP_FIXED is not set, the implementation uses *addr* in an implementation-defined manner to arrive at *pa*. The *pa* so chosen shall be an area of the address space that the implementation deems suitable for a mapping of *len* bytes to the file. All implementations interpret an *addr* value of 0 as granting the implementation complete freedom in selecting *pa*, subject to constraints described below. A non-zero value of *addr* is taken to be a suggestion of a process address near which the mapping should be placed. When the implementation selects a value for *pa*, it never places a mapping at address 0, nor does it replace any extant mapping.

If MAP_FIXED is specified and *addr* is non-zero, it shall have the same remainder as the *off* parameter, modulo the page size as returned by *sysconf*() when passed _SC_PAGESIZE or _SC_PAGE_SIZE. The implementation may require that off is a multiple of the page size. If MAP_FIXED is specified, the implementation may require that *addr* is a multiple of the page size. The system performs mapping operations over whole pages. Thus, while the parameter *len* need not meet a size or alignment constraint, the system shall include, in any mapping operation, any partial page specified by the address range starting at *pa* and continuing for *len* bytes.

The system shall always zero-fill any partial page at the end of an object. Further, the system shall never write out any modified portions of the last page of an object which are beyond its end. References within the address range starting at *pa* and continuing for *len* bytes to whole pages following the end of an object shall result in delivery of a SIGBUS signal.

An implementation may generate SIGBUS signals when a reference would cause an error in the mapped object, such as out-of-space condition.

The *mmap*() function shall add an extra reference to the file associated with the file descriptor *fildes* which is not removed by a subsequent *close*() on that file descriptor. This reference shall be removed when there are no more mappings to the file.

The last data access timestamp of the mapped file may be marked for update at any time between the *mmap*() call and the corresponding *munmap*() call. The initial read or write reference to a mapped region shall cause the file's last data access timestamp to be marked for update if it has not already been marked for update.

The last data modification and last file status change timestamps of a file that is mapped with MAP_SHARED and PROT_WRITE shall be marked for update at some point in the interval between a write reference to the mapped region and the next call to *msync*() with MS_ASYNC or MS_SYNC for that portion of the file by any process. If there is no such call and if the underlying file is modified as a result of a write reference, then these timestamps shall be marked for update at some time after the write reference.

There may be implementation-defined limits on the number of memory regions that can be mapped (per process or per system).

If such a limit is imposed, whether the number of memory regions that can be mapped by a process is decreased by the use of *shmat*() is implementation-defined.

If *mmap*() fails for reasons other than **[EBADF]**, **[EINVAL]**, or **[ENOTSUP]**, some of the mappings in the address range starting at *addr* and continuing for *len* bytes may have been unmapped.

RETURN VALUE

Upon successful completion, the *mmap*() function shall return the address at which the mapping was placed (*pa*); otherwise, it shall return a value of MAP_FAILED and set *errno* to indicate the error. The symbol MAP_FAILED is defined in the <*sys/mman.h*> header. No successful return from *mmap*() shall return the value MAP_FAILED.

ERRORS

The *mmap*() function shall fail if:

EACCES
The *fildes* argument is not open for read, regardless of the protection specified, or *fildes* is not open for write and PROT_WRITE was specified for a MAP_SHARED type mapping.

EAGAIN
The mapping could not be locked in memory, if required by *mlockall*(), due to a lack of resources.

EBADF
The *fildes* argument is not a valid open file descriptor.

EINVAL
The value of *len* is zero.

EINVAL
The value of *flags* is invalid (neither MAP_PRIVATE nor MAP_SHARED is set).

EMFILE
The number of mapped regions would exceed an implementation-defined limit (per process or per system).

ENODEV
The *fildes* argument refers to a file whose type is not supported by *mmap*().

ENOMEM
MAP_FIXED was specified, and the range [*addr*,*addr*+*len*) exceeds that allowed for the address space of a process; or, if MAP_FIXED was not specified and there is insufficient room in the address space to effect the mapping.

ENOMEM
The mapping could not be locked in memory, if required by *mlockall*(), because it would require more space than the system is able to supply.

ENOMEM
Not enough unallocated memory resources remain in the typed memory object designated by *fildes* to allocate *len* bytes.

ENOTSUP
MAP_FIXED or MAP_PRIVATE was specified in the *flags* argument and the implementation does not support this functionality.

The implementation does not support the combination of accesses requested in the *prot* argument.

ENXIO
Addresses in the range [*off*,*off*+*len*) are invalid for the object specified by *fildes*.

ENXIO
MAP_FIXED was specified in *flags* and the combination of *addr*, *len*, and *off* is invalid for the object specified by *fildes*.

ENXIO
The *fildes* argument refers to a typed memory object that is not accessible from the calling process.

EOVERFLOW
The file is a regular file and the value of *off* plus *len* exceeds the offset maximum established in the open file description associated with *fildes*.

The *mmap*() function may fail if:

EINVAL
The *addr* argument (if MAP_FIXED was specified) or *off* is not a multiple of the page size as returned by *sysconf*(), or is considered invalid by the implementation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Use of *mmap*() may reduce the amount of memory available to other memory allocation functions.

Use of MAP_FIXED may result in unspecified behavior in further use of *malloc*() and *shmat*(). The use of MAP_FIXED is discouraged, as it may prevent an implementation from making the most effective use of resources. Most implementations require that *off* and *addr* are multiples of the page size as returned by *sysconf*().

The application must ensure correct synchronization when using *mmap*() in conjunction with any other file access method, such as *read*() and *write*(), standard input/output, and *shmat*().

The *mmap*() function allows access to resources via address space manipulations, instead of *read*()/*write*(). Once a file is mapped, all a process has to do to access it is use the data at the address to which the file was mapped. So, using pseudo-code to illustrate the way in which an existing program might be changed to use *mmap*(), the following:

```
fildes = open(...)
lseek(fildes, some_offset)
read(fildes, buf, len)
/* Use data in buf. */
```

becomes:

```
fildes = open(...)
address = mmap(0, len, PROT_READ, MAP_PRIVATE, fildes, some_offset)
/* Use data at address. */
```

RATIONALE

After considering several other alternatives, it was decided to adopt the *mmap*() definition found in SVR4 for mapping memory objects into process address spaces. The SVR4 definition is minimal, in that it describes only what has been built, and what appears to be necessary for a general and portable mapping facility.

Note that while *mmap*() was first designed for mapping files, it is actually a general-purpose mapping facility. It can be used to map any appropriate object, such as memory, files, devices, and so on, into the address space of a process.

When a mapping is established, it is possible that the implementation may need to map more than is requested into the address space of the process because of hardware requirements. An application, however, cannot count on this behavior. Implementations that do not use a paged architecture may simply allocate a common memory region and return the address of it; such implementations probably do not allocate any more than is necessary. References past the end of the requested area are unspecified.

If an application requests a mapping that overlaps existing mappings in the process, it might be desirable that an implementation detect this and inform the application. However, if the program specifies a fixed address mapping (which requires some implementation knowledge to determine a suitable address, if the function is supported at all), then the program is presumed to be successfully managing its own address space and should be trusted when it asks to map over existing data structures. Furthermore, it is also desirable to make as few system calls as possible, and it might be considered onerous to require an *munmap*() before an *mmap*() to the same address range. This volume of POSIX.1-2017 specifies that the new mapping replaces any existing mappings (implying an automatic *munmap*() on the address range), following existing practice in this regard. The standard developers also considered whether there should be a way for new mappings to overlay existing mappings, but found no existing practice for this.

It is not expected that all hardware implementations are able to support all combinations of permissions at all addresses. Implementations are required to disallow write access to mappings without write permission and to disallow access to mappings without any access permission. Other than these restrictions, implementations may allow access types other than those requested by the application. For example, if the application requests only PROT_WRITE, the implementation may also allow read access. A call to *mmap*() fails if the implementation cannot support allowing all the access requested by the application. For example, some implementations cannot support a request for both write access and execute access simultaneously. All implementations must support requests for no access, read access, write access, and both read and write access. Strictly conforming code must only rely on the required checks. These restrictions allow for portability across a wide range of hardware.

The MAP_FIXED address treatment is likely to fail for non-page-aligned values and for certain architecture-dependent address ranges. Conforming implementations cannot count on being able to choose address values for MAP_FIXED without utilizing non-portable, implementation-defined knowledge. Nonetheless, MAP_FIXED is provided as a standard interface conforming to existing practice for utilizing such knowledge when it is available.

Similarly, in order to allow implementations that do not support virtual addresses, support for directly specifying any mapping addresses via MAP_FIXED is not required and thus a conforming application may not count on it.

The MAP_PRIVATE function can be implemented efficiently when memory protection hardware is available. When such hardware is not available, implementations can implement such "mappings" by simply making a real copy of the relevant data into process private memory, though this tends to behave similarly to *read*().

The function has been defined to allow for many different models of using shared memory. However, all uses are not equally portable across all machine architectures. In particular, the *mmap*() function allows the system as well as the application to specify the address at which to map a specific region of a memory object. The most portable way to use the function is always to let the system choose the address, specifying NULL as the value for the argument *addr* and not to specify MAP_FIXED.

If it is intended that a particular region of a memory object be mapped at the same address in a group of processes (on machines where this is even possible), then MAP_FIXED can be used to pass in the desired mapping address. The system can still be used to choose the desired address if the first such mapping is made without specifying MAP_FIXED, and then the resulting mapping address can be passed to subsequent processes for them to pass in via MAP_FIXED. The availability of a specific address range cannot be guaranteed, in general.

The *mmap*() function can be used to map a region of memory that is larger than the current size of the object. Memory access within the mapping but beyond the current end of the underlying objects may result in SIGBUS signals being sent to the process. The reason for this is that the size of the object can be manipulated by other processes and can change at any moment. The implementation should tell the application that a memory reference is outside the object where this can be detected; otherwise, written data may be lost and read data may not reflect actual data in the object.

Note that references beyond the end of the object do not extend the object as the new end cannot be determined precisely by most virtual memory hardware. Instead, the size can be directly manipulated by *ftruncate*().

Process memory locking does apply to shared memory regions, and the MCL_FUTURE argument to *mlockall*() can be relied upon to cause new shared memory regions to be automatically locked.

Existing implementations of *mmap*() return the value −1 when unsuccessful. Since the casting of this value to type **void *** cannot be guaranteed by the ISO C standard to be distinct from a successful value, this volume of POSIX.1-2017 defines the symbol MAP_FAILED, which a conforming implementation does not return as the result of a successful call.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fcntl*( ), *fork*( ), *lockf*( ), *msync*( ), *munmap*( ), *mprotect*( ), *posix_typed_mem_open*( ), *shmat*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

modf, modff, modfl — decompose a floating-point number

SYNOPSIS

```
#include <math.h>

double modf(double x, double *iptr);
float modff(float value, float *iptr);
long double modfl(long double value, long double *iptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall break the argument *x* into integral and fractional parts, each of which has the same sign as the argument. It stores the integral part as a **double** (for the *modf*() function), a **float** (for the *modff*() function), or a **long double** (for the *modfl*() function), in the object pointed to by *iptr*.

RETURN VALUE

Upon successful completion, these functions shall return the signed fractional part of *x*.

If *x* is NaN, a NaN shall be returned, and **iptr* shall be set to a NaN.

If *x* is ±Inf, ±0 shall be returned, and **iptr* shall be set to ±Inf.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *modf*() function computes the function result and **iptr* such that:

```
a = modf(x, iptr) ;
x == a+*iptr ;
```

allowing for the usual floating-point inaccuracies.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

frexp( ), *isnan*( ), *ldexp*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mprotect — set protection of memory mapping

SYNOPSIS

#include <sys/mman.h>

int mprotect(void **addr*, size_t *len*, int *prot*);

DESCRIPTION

The *mprotect*() function shall change the access protections to be that specified by *prot* for those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. The parameter *prot* determines whether read, write, execute, or some combination of accesses are permitted to the data being mapped. The *prot* argument should be either PROT_NONE or the bitwise-inclusive OR of one or more of PROT_READ, PROT_WRITE, and PROT_EXEC.

If an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect*() shall fail.

An implementation may permit accesses other than those specified by *prot*; however, no implementation shall permit a write to succeed where PROT_WRITE has not been set or shall permit any access where PROT_NONE alone has been set. Implementations shall support at least the following values of *prot*: PROT_NONE, PROT_READ, PROT_WRITE, and the bitwise-inclusive OR of PROT_READ and PROT_WRITE. If PROT_WRITE is specified, the application shall ensure that it has opened the mapped objects in the specified address range with write permission, unless MAP_PRIVATE was specified in the original mapping, regardless of whether the file descriptors used to map the objects have since been closed.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf*().

The behavior of this function is unspecified if the mapping was not established by a call to *mmap*().

When *mprotect*() fails for reasons other than **[EINVAL]**, the protections on some of the pages in the range [*addr*,*addr*+*len*) may have been changed.

RETURN VALUE

Upon successful completion, *mprotect*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *mprotect*() function shall fail if:

EACCES
The *prot* argument specifies a protection that violates the access permission the process has to the underlying memory object.

EAGAIN
The *prot* argument specifies PROT_WRITE over a MAP_PRIVATE mapping and there are insufficient memory resources to reserve for locking the private page.

ENOMEM
Addresses in the range [*addr*,*addr*+*len*) are invalid for the address space of a process, or specify one or more pages which are not mapped.

ENOMEM
The *prot* argument specifies PROT_WRITE on a MAP_PRIVATE mapping, and it would require more space than the system is able to supply for locking the private pages, if required.

ENOTSUP
The implementation does not support the combination of accesses requested in the *prot* argument.

The *mprotect*() function may fail if:

EINVAL
The *addr* argument is not a multiple of the page size as returned by *sysconf*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Most implementations require that *addr* is a multiple of the page size as returned by *sysconf*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_close — close a message queue (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

int mq_close(mqd_t mqdes);
```

DESCRIPTION

The *mq_close*() function shall remove the association between the message queue descriptor, *mqdes*, and its message queue. The results of using this message queue descriptor after successful return from this *mq_close*(), and until the return of this message queue descriptor from a subsequent *mq_open*(), are undefined.

If the process has successfully attached a notification request to the message queue via this *mqdes*, this attachment shall be removed, and the message queue is available for another process to attach for notification.

RETURN VALUE

Upon successful completion, the *mq_close*() function shall return a value of zero; otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_close*() function shall fail if:

EBADF

The *mqdes* argument is not a valid message queue descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_getattr — get message queue attributes (**REALTIME**)

SYNOPSIS

#include <mqueue.h>

int mq_getattr(mqd_t *mqdes*, struct mq_attr **mqstat*);

DESCRIPTION

The *mq_getattr*() function shall obtain status information and attributes of the message queue and the open message queue description associated with the message queue descriptor.

The *mqdes* argument specifies a message queue descriptor.

The results shall be returned in the **mq_attr** structure referenced by the *mqstat* argument.

Upon return, the following members shall have the values associated with the open message queue description as set when the message queue was opened and as modified by subsequent *mq_setattr*() calls: *mq_flags*.

The following attributes of the message queue shall be returned as set at message queue creation: *mq_maxmsg*, *mq_msgsize*.

Upon return, the following members within the **mq_attr** structure referenced by the *mqstat* argument shall be set to the current state of the message queue:

mq_curmsgs
The number of messages currently on the queue.

RETURN VALUE

Upon successful completion, the *mq_getattr*() function shall return zero. Otherwise, the function shall return −1 and set *errno* to indicate the error.

ERRORS

The *mq_getattr*() function may fail if:

EBADF
The *mqdes* argument is not a valid message queue descriptor.

The following sections are informative.

EXAMPLES

See *mq_notify*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_notify( ), *mq_open*( ), *mq_send*( ), *mq_setattr*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_notify — notify process that a message is available (**REALTIME**)

SYNOPSIS

#include <mqueue.h>

int mq_notify(mqd_t *mqdes*, const struct sigevent **notification*);

DESCRIPTION

If the argument *notification* is not NULL, this function shall register the calling process to be notified of message arrival at an empty message queue associated with the specified message queue descriptor, *mqdes*. The notification specified by the *notification* argument shall be sent to the process when the message queue transitions from empty to non-empty. At any time, only one process may be registered for notification by a message queue. If the calling process or any other process has already registered for notification of message arrival at the specified message queue, subsequent attempts to register for that message queue shall fail.

If *notification* is NULL and the process is currently registered for notification by the specified message queue, the existing registration shall be removed.

When the notification is sent to the registered process, its registration shall be removed. The message queue shall then be available for registration.

If a process has registered for notification of message arrival at a message queue and some thread is blocked in *mq_receive*() or *mq_timedreceive*() waiting to receive a message when a message arrives at the queue, the arriving message shall satisfy the appropriate *mq_receive*() or *mq_timedreceive*(), respectively. The resulting behavior is as if the message queue remains empty, and no notification shall be sent.

RETURN VALUE

Upon successful completion, the *mq_notify*() function shall return a value of zero; otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_notify*() function shall fail if:

EBADF
The *mqdes* argument is not a valid message queue descriptor.

EBUSY
A process is already registered for notification by the message queue.

The *mq_notify*() function may fail if:

EINVAL
The *notification* argument is NULL and the process is currently not registered.

The following sections are informative.

EXAMPLES

The following program registers a notification request for the message queue named in its command-line argument. Notification is performed by creating a thread. The thread executes a function which reads one message from the queue and then terminates the process.

```
#include <pthread.h>
#include <mqueue.h>
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <unistd.h>

static void                     /* Thread start function */
tfunc(union sigval sv)
{
    struct mq_attr attr;
    ssize_t nr;
    void *buf;
    mqd_t mqdes = *((mqd_t *) sv.sival_ptr);

    /* Determine maximum msg size; allocate buffer to receive msg */

    if (mq_getattr(mqdes, &attr) == -1) {
        perror("mq_getattr");
        exit(EXIT_FAILURE);
    }
    buf = malloc(attr.mq_msgsize);

    if (buf == NULL) {
        perror("malloc");
        exit(EXIT_FAILURE);
    }

    nr = mq_receive(mqdes, buf, attr.mq_msgsize, NULL);
    if (nr == -1) {
        perror("mq_receive");
        exit(EXIT_FAILURE);
    }

    printf("Read %ld bytes from message queue\n", (long) nr);
    free(buf);
    exit(EXIT_SUCCESS);         /* Terminate the process */
}

int
main(int argc, char *argv[])
{
    mqd_t mqdes;
    struct sigevent not;

    assert(argc == 2);

    mqdes = mq_open(argv[1], O_RDONLY);
    if (mqdes == (mqd_t) -1) {
        perror("mq_open");
        exit(EXIT_FAILURE);
    }

    not.sigev_notify = SIGEV_THREAD;
    not.sigev_notify_function = tfunc;
    not.sigev_notify_attributes = NULL;
    not.sigev_value.sival_ptr = &mqdes;   /* Arg. to thread func. */
    if (mq_notify(mqdes, &not) == -1) {
        perror("mq_notify");
        exit(EXIT_FAILURE);
    }

    pause();    /* Process will be terminated by thread function */
}
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *mq_receive*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_open — open a message queue (**REALTIME**)

SYNOPSIS

#include <mqueue.h>

mqd_t mq_open(const char **name*, int *oflag*, ...);

DESCRIPTION

The *mq_open*() function shall establish the connection between a process and a message queue with a message queue descriptor. It shall create an open message queue description that refers to the message queue, and a message queue descriptor that refers to that open message queue description. The message queue descriptor is used by other functions to refer to that message queue. The *name* argument points to a string naming a message queue. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *mq_open*() with the same value of *name* shall refer to the same message queue object, as long as that name has not been removed. If *name* does not begin with the <slash> character, the effect is implementation-defined. If the *name* argument is not the name of an existing message queue and creation is not requested, *mq_open*() shall fail and return an error.

A message queue descriptor may be implemented using a file descriptor, in which case applications can open up to at least {OPEN_MAX} file and message queues.

The *oflag* argument requests the desired receive and/or send access to the message queue. The requested access permission to receive messages or send messages shall be granted if the calling process would be granted read or write access, respectively, to an equivalently protected file.

The value of *oflag* is the bitwise-inclusive OR of values from the following list. Applications shall specify exactly one of the first three values (access modes) below in the value of *oflag*:

O_RDONLY Open the message queue for receiving messages. The process can use the returned message queue descriptor with *mq_receive*(), but not *mq_send*(). A message queue may be open multiple times in the same or different processes for receiving messages.

O_WRONLY Open the queue for sending messages. The process can use the returned message queue descriptor with *mq_send*() but not *mq_receive*(). A message queue may be open multiple times in the same or different processes for sending messages.

O_RDWR Open the queue for both receiving and sending messages. The process can use any of the functions allowed for O_RDONLY and O_WRONLY. A message queue may be open multiple times in the same or different processes for sending messages.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT Create a message queue. It requires two additional arguments: *mode*, which shall be of type **mode_t**, and *attr*, which shall be a pointer to an **mq_attr** structure. If the pathname *name* has already been used to create a message queue that still exists, then this flag shall have no effect, except as noted under O_EXCL. Otherwise, a message queue shall be created without any messages in it. The user ID of the message queue shall be set to the effective user ID of the process. The group ID of the message queue shall be set to the effective group ID of the process; however, if the *name* argument is visible in the file system, the group ID may be set to the group ID of the containing directory. When bits in *mode* other than the file permission bits are specified, the effect is unspecified. If *attr* is NULL, the message queue shall be created with implementation-defined default message queue attributes. If *attr* is non-NULL and the calling process has appropriate privileges on *name*, the message queue *mq_maxmsg* and *mq_msgsize* attributes shall be set to the values of the corresponding members in the **mq_attr** structure referred to by *attr*. The values of the *mq_flags* and *mq_curmsgs* members of the **mq_attr** structure shall be ignored. If *attr* is non-NULL, but the calling process does not have appropriate privileges on *name*, the *mq_open*() function shall fail and return an error without creating the message queue.

O_EXCL
If O_EXCL and O_CREAT are set, *mq_open*() shall fail if the message queue *name* exists. The check for the existence of the message queue and the creation of the message queue if it does not exist shall be atomic with respect to other threads executing *mq_open*() naming the same *name* with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NONBLOCK
Determines whether an *mq_send*() or *mq_receive*() waits for resources or messages that are not currently available, or fails with *errno* set to **[EAGAIN]**; see *mq_send*( ) and *mq_receive*( ) for details.

The *mq_open*() function does not add or remove messages from the queue.

RETURN VALUE

Upon successful completion, the function shall return a message queue descriptor; otherwise, the function shall return (**mqd_t**)−1 and set *errno* to indicate the error.

ERRORS

The *mq_open*() function shall fail if:

EACCES
The message queue exists and the permissions specified by *oflag* are denied, or the message queue does not exist and permission to create the message queue is denied.

EEXIST
O_CREAT and O_EXCL are set and the named message queue already exists.

EINTR
The *mq_open*() function was interrupted by a signal.

EINVAL
The *mq_open*() function is not supported for the given name.

EINVAL
O_CREAT was specified in *oflag*, the value of *attr* is not NULL, and either *mq_maxmsg* or *mq_msgsize* was less than or equal to zero.

EMFILE
Too many message queue descriptors or file descriptors are currently in use by this process.

ENFILE
Too many message queues are currently open in the system.

ENOENT
O_CREAT is not set and the named message queue does not exist.

ENOSPC
There is insufficient space for the creation of the new message queue.

If any of the following conditions occur, the *mq_open*() function may return (**mqd_t**)−1 and set *errno* to the corresponding value.

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *mq_open*() and *mq_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

mq_close( ), *mq_getattr*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_receive, mq_timedreceive — receive a message from a message queue (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

ssize_t mq_receive(mqd_t mqdes, char *msg_ptr, size_t msg_len,
   unsigned *msg_prio);

#include <mqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
   size_t msg_len, unsigned *restrict msg_prio,
   const struct timespec *restrict abstime);
```

DESCRIPTION

The *mq_receive*() function shall receive the oldest of the highest priority message(s) from the message queue specified by *mqdes*. If the size of the buffer in bytes, specified by the *msg_len* argument, is less than the *mq_msgsize* attribute of the message queue, the function shall fail and return an error. Otherwise, the selected message shall be removed from the queue and copied to the buffer pointed to by the *msg_ptr* argument.

If the value of *msg_len* is greater than {SSIZE_MAX}, the result is implementation-defined.

If the argument *msg_prio* is not NULL, the priority of the selected message shall be stored in the location referenced by *msg_prio*.

If the specified message queue is empty and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_receive*() shall block until a message is enqueued on the message queue or until *mq_receive*() is interrupted by a signal. If more than one thread is waiting to receive a message when a message arrives at an empty queue and the Priority Scheduling option is supported, then the thread of highest priority that has been waiting the longest shall be selected to receive the message. Otherwise, it is unspecified which waiting thread receives the message. If the specified message queue is empty and O_NONBLOCK is set in the message queue description associated with *mqdes*, no message shall be removed from the queue, and *mq_receive*() shall return an error.

The *mq_timedreceive*() function shall receive the oldest of the highest priority messages from the message queue specified by *mqdes* as described for the *mq_receive*() function. However, if O_NONBLOCK was not specified when the message queue was opened via the *mq_open*() function, and no message exists on the queue to satisfy the receive, the wait for such a message shall be terminated when the specified timeout expires. If O_NONBLOCK is set, this function is equivalent to *mq_receive*().

The timeout expires when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The *timespec* argument is defined in the *<time.h>* header.

Under no circumstance shall the operation fail with a timeout if a message can be removed from the message queue immediately. The validity of the *abstime* parameter need not be checked if a message can be removed from the message queue immediately.

RETURN VALUE

Upon successful completion, the *mq_receive*() and *mq_timedreceive*() functions shall return the length of the selected message in bytes and the message shall be removed from the queue. Otherwise, no message shall be removed from the queue, the functions shall return a value of −1, and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EAGAIN
O_NONBLOCK was set in the message description associated with *mqdes*, and the specified message queue is empty.

EBADF
The *mqdes* argument is not a valid message queue descriptor open for reading.

EMSGSIZE
The specified message buffer size, *msg_len*, is less than the message size attribute of the message queue.

EINTR
The *mq_receive*() or *mq_timedreceive*() operation was interrupted by a signal.

EINVAL
The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

ETIMEDOUT
The O_NONBLOCK flag was not set when the message queue was opened, but no message arrived on the queue before the specified timeout expired.

These functions may fail if:

EBADMSG
The implementation has detected a data corruption problem with the message.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_send, mq_timedsend — send a message to a message queue (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

int mq_send(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
   unsigned msg_prio);

#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
   unsigned msg_prio, const struct timespec *abstime);
```

DESCRIPTION

The *mq_send*() function shall add the message pointed to by the argument *msg_ptr* to the message queue specified by *mqdes*. The *msg_len* argument specifies the length of the message, in bytes, pointed to by *msg_ptr*. The value of *msg_len* shall be less than or equal to the *mq_msgsize* attribute of the message queue, or *mq_send*() shall fail.

If the specified message queue is not full, *mq_send*() shall behave as if the message is inserted into the message queue at the position indicated by the *msg_prio* argument. A message with a larger numeric value of *msg_prio* shall be inserted before messages with lower values of *msg_prio*. A message shall be inserted after other messages in the queue, if any, with equal *msg_prio*. The value of *msg_prio* shall be less than {MQ_PRIO_MAX}.

If the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, *mq_send*() shall block until space becomes available to enqueue the message, or until *mq_send*() is interrupted by a signal. If more than one thread is waiting to send when space becomes available in the message queue and the Priority Scheduling option is supported, then the thread of the highest priority that has been waiting the longest shall be unblocked to send its message. Otherwise, it is unspecified which waiting thread is unblocked. If the specified message queue is full and O_NONBLOCK is set in the message queue description associated with *mqdes*, the message shall not be queued and *mq_send*() shall return an error.

The *mq_timedsend*() function shall add a message to the message queue specified by *mqdes* in the manner defined for the *mq_send*() function. However, if the specified message queue is full and O_NONBLOCK is not set in the message queue description associated with *mqdes*, the wait for sufficient room in the queue shall be terminated when the specified timeout expires. If O_NONBLOCK is set in the message queue description, this function shall be equivalent to *mq_send*().

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The *timespec* argument is defined in the *<time.h>* header.

Under no circumstance shall the operation fail with a timeout if there is sufficient room in the queue to add the message immediately. The validity of the *abstime* parameter need not be checked when there is sufficient room in the queue.

RETURN VALUE

Upon successful completion, the *mq_send*() and *mq_timedsend*() functions shall return a value of zero. Otherwise, no message shall be enqueued, the functions shall return −1, and *errno* shall be set to indicate the error.

ERRORS

The *mq_send*() and *mq_timedsend*() functions shall fail if:

EAGAIN
The O_NONBLOCK flag is set in the message queue description associated with *mqdes*, and the specified message queue is full.

EBADF
The *mqdes* argument is not a valid message queue descriptor open for writing.

EINTR
A signal interrupted the call to *mq_send*() or *mq_timedsend*().

EINVAL
The value of *msg_prio* was outside the valid range.

EINVAL
The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

EMSGSIZE
The specified message length, *msg_len*, exceeds the message size attribute of the message queue.

ETIMEDOUT
The O_NONBLOCK flag was not set when the message queue was opened, but the timeout expired before the message could be added to the queue.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The value of the symbol {MQ_PRIO_MAX} limits the number of priority levels supported by the application. Message priorities range from 0 to {MQ_PRIO_MAX}−1.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_receive*( ), *mq_setattr*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_setattr — set message queue attributes (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

int mq_setattr(mqd_t mqdes, const struct mq_attr *restrict mqstat,
   struct mq_attr *restrict omqstat);
```

DESCRIPTION

The *mq_setattr*() function shall set attributes associated with the open message queue description referenced by the message queue descriptor specified by *mqdes*.

The message queue attributes corresponding to the following members defined in the **mq_attr** structure shall be set to the specified values upon successful completion of *mq_setattr*():

mq_flags The value of this member is the bitwise-logical OR of zero or more of O_NONBLOCK and any implementation-defined flags.

The values of the *mq_maxmsg*, *mq_msgsize*, and *mq_curmsgs* members of the **mq_attr** structure shall be ignored by *mq_setattr*().

If *omqstat* is non-NULL, the *mq_setattr*() function shall store, in the location referenced by *omqstat*, the previous message queue attributes and the current queue status. These values shall be the same as would be returned by a call to *mq_getattr*() at that point.

RETURN VALUE

Upon successful completion, the function shall return a value of zero and the attributes of the message queue shall have been changed as specified.

Otherwise, the message queue attributes shall be unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_setattr*() function shall fail if:

EBADF
The *mqdes* argument is not a valid message queue descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mq_open( ), *mq_send*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_timedreceive — receive a message from a message queue (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <mqueue.h>
#include <time.h>

ssize_t mq_timedreceive(mqd_t mqdes, char *restrict msg_ptr,
    size_t msg_len, unsigned *restrict msg_prio,
    const struct timespec *restrict abstime);
```

DESCRIPTION

Refer to *mq_receive*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_timedsend — send a message to a message queue (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <mqueue.h>
#include <time.h>

int mq_timedsend(mqd_t mqdes, const char *msg_ptr, size_t msg_len,
   unsigned msg_prio, const struct timespec *abstime);
```

DESCRIPTION

Refer to *mq_send*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mq_unlink — remove a message queue (**REALTIME**)

SYNOPSIS

```
#include <mqueue.h>

int mq_unlink(const char *name);
```

DESCRIPTION

The *mq_unlink*() function shall remove the message queue named by the string name. If one or more processes have the message queue open when *mq_unlink*() is called, destruction of the message queue shall be postponed until all references to the message queue have been closed. However, the *mq_unlink*() call need not block until all references have been closed; it may return immediately.

After a successful call to *mq_unlink*(), reuse of the name shall subsequently cause *mq_open*() to behave as if no message queue of this name exists (that is, *mq_open*() will fail if O_CREAT is not set, or will create a new message queue if O_CREAT is set).

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the named message queue shall be unchanged by this function call, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *mq_unlink*() function shall fail if:

EACCES
Permission is denied to unlink the named message queue.

EINTR
The call to *mq_unlink*() blocked waiting for all references to the named message queue to be closed and a signal interrupted the call.

ENOENT
The named message queue does not exist.

The *mq_unlink*() function may fail if:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *mq_unlink*() with a *name* argument that contains the same message queue name as was previously used in a successful *mq_open*() call shall not give an **[ENAMETOOLONG]** error.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *mq_open*() and *mq_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

mq_close( ), *mq_open*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, **<mqueue.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

mrand48 — generate uniformly distributed pseudo-random signed long integers

SYNOPSIS

```
#include <stdlib.h>

long mrand48(void);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

msgctl — XSI message control operations

SYNOPSIS

```
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

DESCRIPTION

The *msgctl*() function operates on XSI message queues (see the Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *msgctl*() function shall provide message control operations as specified by *cmd*. The following values for *cmd*, and the message control operations they specify, are:

IPC_STAT Place the current value of each member of the **msqid_ds** data structure associated with *msqid* into the structure pointed to by *buf*. The contents of this structure are defined in *<sys/msg.h>*.

IPC_SET Set the value of the following members of the **msqid_ds** data structure associated with *msqid* to the corresponding value found in the structure pointed to by *buf*:

```
msg_perm.uid
msg_perm.gid
msg_perm.mode
msg_qbytes
```

Also, the *msg_ctime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

IPC_SET can only be executed by a process with appropriate privileges or that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*. Only a process with appropriate privileges can raise the value of **msg_qbytes**.

IPC_RMID Remove the message queue identifier specified by *msqid* from the system and destroy the message queue and **msqid_ds** data structure associated with it. IPC_RMD can only be executed by a process with appropriate privileges or one that has an effective user ID equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the **msqid_ds** data structure associated with *msqid*.

RETURN VALUE

Upon successful completion, *msgctl*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *msgctl*() function shall fail if:

EACCES
The argument *cmd* is IPC_STAT and the calling process does not have read permission; see *Section 2.7*, *XSI Interprocess Communication*.

EINVAL
The value of *msqid* is not a valid message queue identifier; or the value of *cmd* is not a valid command.

EPERM

The argument *cmd* is IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of **msg_perm.cuid** or **msg_perm.uid** in the data structure associated with *msqid*.

EPERM

The argument *cmd* is IPC_SET, an attempt is being made to increase to the value of **msg_qbytes**, and the effective user ID of the calling process does not have appropriate privileges.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgget*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*, **<sys_msg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

msgget — get the XSI message queue identifier

SYNOPSIS

```
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

DESCRIPTION

The *msgget*() function operates on XSI message queues (see the Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *msgget*() function shall return the message queue identifier associated with the argument *key*.

A message queue identifier, associated message queue, and data structure (see *<sys/msg.h>*), shall be created for the argument *key* if one of the following is true:

* The argument *key* is equal to IPC_PRIVATE.
* The argument *key* does not already have a message queue identifier associated with it, and (*msgflg* & IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new message queue identifier shall be initialized as follows:

* **msg_perm.cuid**, **msg_perm.uid**, **msg_perm.cgid**, and **msg_perm.gid** shall be set to the effective user ID and effective group ID, respectively, of the calling process.
* The low-order 9 bits of **msg_perm.mode** shall be set to the low-order 9 bits of *msgflg*.
* **msg_qnum**, **msg_lspid**, **msg_lrpid**, **msg_stime**, and **msg_rtime** shall be set to 0.
* **msg_ctime** shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.
* **msg_qbytes** shall be set to the system limit.

RETURN VALUE

Upon successful completion, *msgget*() shall return a non-negative integer, namely a message queue identifier. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *msgget*() function shall fail if:

EACCES
A message queue identifier exists for the argument *key*, but operation permission as specified by the low-order 9 bits of *msgflg* would not be granted; see *Section 2.7*, *XSI Interprocess Communication*.

EEXIST
A message queue identifier exists for the argument *key* but ((*msgflg* & IPC_CREAT) && (*msgflg* & IPC_EXCL)) is non-zero.

ENOENT
A message queue identifier does not exist for the argument *key* and (*msgflg* & IPC_CREAT) is 0.

ENOSPC
A message queue identifier is to be created but the system-imposed limit on the maximum number of allowed message queue identifiers system-wide would be exceeded.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *ftok*( ), *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgrcv*( ), *msgsnd*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*, **<sys_msg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

msgrcv — XSI message receive operation

SYNOPSIS

```
#include <sys/msg.h>

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,
   int msgflg);
```

DESCRIPTION

The *msgrcv*() function operates on XSI message queues (see the Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *msgrcv*() function shall read a message from the queue associated with the message queue identifier specified by *msqid* and place it in the user-defined buffer pointed to by *msgp*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
   long   mtype;    /* Message type. */
   char   mtext[1]; /* Message text. */
}
```

The structure member *mtype* is the received message's type as specified by the sending process.

The structure member *mtext* is the text of the message.

The argument *msgsz* specifies the size in bytes of *mtext*. The received message shall be truncated to *msgsz* bytes if it is larger than *msgsz* and (*msgflg* & MSG_NOERROR) is non-zero. The truncated part of the message shall be lost and no indication of the truncation shall be given to the calling process.

If the value of *msgsz* is greater than {SSIZE_MAX}, the result is implementation-defined.

The argument *msgtyp* specifies the type of message requested as follows:

* If *msgtyp* is 0, the first message on the queue shall be received.
* If *msgtyp* is greater than 0, the first message of type *msgtyp* shall be received.
* If *msgtyp* is less than 0, the first message of the lowest type that is less than or equal to the absolute value of *msgtyp* shall be received.

The argument *msgflg* specifies the action to be taken if a message of the desired type is not on the queue. These are as follows:

* If (*msgflg* & IPC_NOWAIT) is non-zero, the calling thread shall return immediately with a return value of −1 and *errno* set to **[ENOMSG]**.
* If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  -- A message of the desired type is placed on the queue.
  -- The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set to **[EIDRM]** and −1 shall be returned.

-- The calling thread receives a signal that is to be caught; in this case a message is not received and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*:

* **msg_qnum** shall be decremented by 1.
* **msg_lrpid** shall be set to the process ID of the calling process.
* **msg_rtime** shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

RETURN VALUE

Upon successful completion, *msgrcv*() shall return a value equal to the number of bytes actually placed into the buffer *mtext*. Otherwise, no message shall be received, *msgrcv*() shall return −1, and *errno* shall be set to indicate the error.

ERRORS

The *msgrcv*() function shall fail if:

E2BIG The value of *mtext* is greater than *msgsz* and (*msgflg* & MSG_NOERROR) is 0.

EACCES
Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EIDRM
The message queue identifier *msqid* is removed from the system.

EINTR
The *msgrcv*() function was interrupted by a signal.

EINVAL
msqid is not a valid message queue identifier.

ENOMSG
The queue does not contain a message of the desired type and (*msgflg* & IPC_NOWAIT) is non-zero.

The following sections are informative.

EXAMPLES

Receiving a Message

The following example receives the first message on the queue (based on the value of the *msgtyp* argument, 0). The queue is identified by the *msqid* argument (assuming that the value has previously been set). This call specifies that an error should be reported if no message is available, but not if the message is too large. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
   long type;
   char text[20];
} msg;
long msgtyp = 0;
...
result = msgrcv(msqid, (void *) &msg, sizeof(msg.text),
        msgtyp, MSG_NOERROR | IPC_NOWAIT);
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgsnd*( ), *sigaction*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*, **<sys_msg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

msgsnd — XSI message send operation

SYNOPSIS

```
#include <sys/msg.h>

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);
```

DESCRIPTION

The *msgsnd*() function operates on XSI message queues (see the Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *msgsnd*() function shall send a message to the queue associated with the message queue identifier specified by *msqid*.

The application shall ensure that the argument *msgp* points to a user-defined buffer that contains first a field of type **long** specifying the type of the message, and then a data portion that holds the data bytes of the message. The structure below is an example of what this user-defined buffer might look like:

```
struct mymsg {
    long   mtype;       /* Message type. */
    char   mtext[1];    /* Message text. */
}
```

The structure member *mtype* is a non-zero positive type **long** that can be used by the receiving process for message selection.

The structure member *mtext* is any text of length *msgsz* bytes. The argument *msgsz* can range from 0 to a system-imposed maximum.

The argument *msgflg* specifies the action to be taken if one or more of the following is true:

* The number of bytes already on the queue is equal to **msg_qbytes**; see *<sys/msg.h>*.
* The total number of messages on all queues system-wide is equal to the system-imposed limit.

These actions are as follows:

* If (*msgflg* & IPC_NOWAIT) is non-zero, the message shall not be sent and the calling thread shall return immediately.
* If (*msgflg* & IPC_NOWAIT) is 0, the calling thread shall suspend execution until one of the following occurs:
  - The condition responsible for the suspension no longer exists, in which case the message is sent.
  - The message queue identifier *msqid* is removed from the system; when this occurs, *errno* shall be set to **[EIDRM]** and −1 shall be returned.
  - The calling thread receives a signal that is to be caught; in this case the message is not sent and the calling thread resumes execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the following actions are taken with respect to the data structure associated with *msqid*; see *<sys/msg.h>*:

* **msg_qnum** shall be incremented by 1.
* **msg_lspid** shall be set to the process ID of the calling process.

* **msg_stime** shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

RETURN VALUE

Upon successful completion, *msgsnd*() shall return 0; otherwise, no message shall be sent, *msgsnd*() shall return −1, and *errno* shall be set to indicate the error.

ERRORS

The *msgsnd*() function shall fail if:

EACCES
Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EAGAIN
The message cannot be sent for one of the reasons cited above and (*msgflg* & IPC_NOWAIT) is non-zero.

EIDRM
The message queue identifier *msqid* is removed from the system.

EINTR
The *msgsnd*() function was interrupted by a signal.

EINVAL
The value of *msqid* is not a valid message queue identifier, or the value of *mtype* is less than 1; or the value of *msgsz* is greater than the system-imposed limit.

The following sections are informative.

EXAMPLES

Sending a Message

The following example sends a message to the queue identified by the *msqid* argument (assuming that value has previously been set). This call specifies that an error should be reported if no message is available. The message size is calculated directly using the *sizeof* operator.

```
#include <sys/msg.h>
...
int result;
int msqid;
struct message {
   long type;
   char text[20];
} msg;

msg.type = 1;
strcpy(msg.text, "This is message 1");
...
result = msgsnd(msqid, (void *) &msg, sizeof(msg.text), IPC_NOWAIT);
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication (IPC). Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *mq_close*( ), *mq_getattr*( ), *mq_notify*( ), *mq_open*( ), *mq_receive*( ), *mq_send*( ), *mq_setattr*( ), *mq_unlink*( ), *msgctl*( ), *msgget*( ), *msgrcv*( ), *sigaction*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.226*, *Message Queue*, **<sys_msg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

msync — synchronize memory with physical storage

SYNOPSIS

```
#include <sys/mman.h>

int msync(void *addr, size_t len, int flags);
```

DESCRIPTION

The *msync*() function shall write all modified data to permanent storage locations, if any, in those whole pages containing any part of the address space of the process starting at address *addr* and continuing for *len* bytes. If no such storage exists, *msync*() need not have any effect. If requested, the *msync*() function shall then invalidate cached copies of data.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf*().

For mappings to files, the *msync*() function shall ensure that all write operations are completed as defined for synchronized I/O data integrity completion. It is unspecified whether the implementation also writes out other file attributes. When the *msync*() function is called on MAP_PRIVATE mappings, any modified data shall not be written to the underlying object and shall not cause such data to be made visible to other processes. It is unspecified whether data in MAP_PRIVATE mappings has any permanent storage locations. The effect of *msync*() on a shared memory object or a typed memory object is unspecified. The behavior of this function is unspecified if the mapping was not established by a call to *mmap*().

The *flags* argument is constructed from the bitwise-inclusive OR of one or more of the following flags defined in the *<sys/mman.h>* header:

center box tab(!); cB | cB lw(1.5i) | lw(2i). Symbolic Constant!Description _ MS_ASYNC!Perform asynchronous writes. MS_SYNC!Perform synchronous writes. MS_INVALIDATE!Invalidate cached data.

When MS_ASYNC is specified, *msync*() shall return immediately once all the write operations are initiated or queued for servicing; when MS_SYNC is specified, *msync*() shall not return until all write operations are completed as defined for synchronized I/O data integrity completion. Either MS_ASYNC or MS_SYNC shall be specified, but not both.

When MS_INVALIDATE is specified, *msync*() shall invalidate all cached copies of mapped data that are inconsistent with the permanent storage locations such that subsequent references shall obtain data that was consistent with the permanent storage locations sometime between the call to *msync*() and the first subsequent memory reference to the data.

If *msync*() causes any write to a file, the file's last data modification and last file status change timestamps shall be marked for update.

RETURN VALUE

Upon successful completion, *msync*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *msync*() function shall fail if:

EBUSY
Some or all of the addresses in the range starting at *addr* and continuing for *len* bytes are locked, and MS_INVALIDATE is specified.

EINVAL
The value of *flags* is invalid.

ENOMEM
The addresses in the range starting at *addr* and continuing for *len* bytes are outside the range allowed for the address space of a process or specify one or more pages that are not mapped.

The *msync*() function may fail if:

EINVAL
The value of *addr* is not a multiple of the page size as returned by *sysconf*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *msync*() function is only supported if the Synchronized Input and Output option is supported, and thus need not be available on all implementations.

The *msync*() function should be used by programs that require a memory object to be in a known state; for example, in building transaction facilities.

Normal system activity can cause pages to be written to disk. Therefore, there are no guarantees that *msync*() is the only control over when pages are or are not written to disk.

RATIONALE

The *msync*() function writes out data in a mapped region to the permanent storage for the underlying object. The call to *msync*() ensures data integrity of the file.

After the data is written out, any cached data may be invalidated if the MS_INVALIDATE flag was specified. This is useful on systems that do not support read/write consistency.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

munlock — unlock a range of process address space

SYNOPSIS

```
#include <sys/mman.h>

int munlock(const void *addr, size_t len);
```

DESCRIPTION

Refer to *mlock*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

munlockall — unlock the address space of a process

SYNOPSIS

```
#include <sys/mman.h>

int munlockall(void);
```

DESCRIPTION

Refer to *mlockall*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

munmap — unmap pages of memory

SYNOPSIS

#include <sys/mman.h>

int munmap(void **addr*, size_t *len*);

DESCRIPTION

The *munmap*() function shall remove any mappings for those entire pages containing any part of the address space of the process starting at *addr* and continuing for *len* bytes. Further references to these pages shall result in the generation of a SIGSEGV signal to the process. If there are no mappings in the specified address range, then *munmap*() has no effect.

The implementation may require that *addr* be a multiple of the page size as returned by *sysconf*().

If a mapping to be removed was private, any modifications made in this address range shall be discarded.

Any memory locks (see *mlock*( ) and *mlockall*( )) associated with this address range shall be removed, as if by an appropriate call to *munlock*().

If a mapping removed from a typed memory object causes the corresponding address range of the memory pool to be inaccessible by any process in the system except through allocatable mappings (that is, mappings of typed memory objects opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag), then that range of the memory pool shall become deallocated and may become available to satisfy future typed memory allocation requests.

A mapping removed from a typed memory object opened with the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag shall not affect in any way the availability of that typed memory for allocation.

The behavior of this function is unspecified if the mapping was not established by a call to *mmap*().

RETURN VALUE

Upon successful completion, *munmap*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *munmap*() function shall fail if:

EINVAL
Addresses in the range [*addr*,*addr*+*len*) are outside the valid range for the address space of a process.

EINVAL
The *len* argument is 0.

The *munmap*() function may fail if:

EINVAL
The *addr* argument is not a multiple of the page size as returned by *sysconf*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *munmap*() function corresponds to SVR4, just as the *mmap*() function does.

It is possible that an application has applied process memory locking to a region that contains shared memory. If this has occurred, the *munmap*() call ignores those locks and, if necessary, causes those locks to be removed.

Most implementations require that *addr* is a multiple of the page size as returned by *sysconf*().

FUTURE DIRECTIONS

None.

SEE ALSO

mlock( ), *mlockall*( ), *mmap*( ), *posix_typed_mem_open*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nan, nanf, nanl — return quiet NaN

SYNOPSIS

```
#include <math.h>

double nan(const char *tagp);
float nanf(const char *tagp);
long double nanl(const char *tagp);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The function call *nan*("*n-char-sequence*") shall be equivalent to:

```
strtod("NAN(n-char-sequence)", (char **) NULL);
```

The function call *nan*(" ") shall be equivalent to:

```
strtod("NAN()", (char **) NULL)
```

If *tagp* does not point to an *n*-**char** sequence or an empty string, the function call shall be equivalent to:

```
strtod("NAN", (char **) NULL)
```

Function calls to *nanf*() and *nanl*() are equivalent to the corresponding function calls to *strtof*() and *strtold*().

RETURN VALUE

These functions shall return a quiet NaN, if available, with content indicated through *tagp*.

If the implementation does not support quiet NaNs, these functions shall return zero.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtod( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nanosleep — high resolution sleep

SYNOPSIS

#include <time.h>

int nanosleep(const struct timespec **rqtp*, struct timespec **rmtp*);

DESCRIPTION

The *nanosleep*() function shall cause the current thread to be suspended from execution until either the time interval specified by the *rqtp* argument has elapsed or a signal is delivered to the calling thread, and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested because the argument value is rounded up to an integer multiple of the sleep resolution or because of the scheduling of other activity by the system. But, except for the case of being interrupted by a signal, the suspension time shall not be less than the time specified by *rqtp*, as measured by the system clock CLOCK_REALTIME.

The use of the *nanosleep*() function has no effect on the action or blockage of any signal.

RETURN VALUE

If the *nanosleep*() function returns because the requested time has elapsed, its return value shall be zero.

If the *nanosleep*() function returns because it has been interrupted by a signal, it shall return a value of −1 and set *errno* to indicate the interruption. If the *rmtp* argument is non-NULL, the **timespec** structure referenced by it is updated to contain the amount of time remaining in the interval (the requested time minus the time actually slept). The *rqtp* and *rmtp* arguments can point to the same object. If the *rmtp* argument is NULL, the remaining time is not returned.

If *nanosleep*() fails, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *nanosleep*() function shall fail if:

EINTR
The *nanosleep*() function was interrupted by a signal.

EINVAL
The *rqtp* argument specified a nanosecond value less than zero or greater than or equal to 1 000 million.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

It is common to suspend execution of a thread for an interval in order to poll the status of a non-interrupting function. A large number of actual needs can be met with a simple extension to *sleep*() that provides finer resolution.

In the POSIX.1-1990 standard and SVR4, it is possible to implement such a routine, but the frequency of wakeup is limited by the resolution of the *alarm*() and *sleep*() functions. In 4.3 BSD, it is possible to write such a routine using no static storage and reserving no system facilities. Although it is possible to write a function with similar functionality to *sleep*() using the remainder of the *timer_**( ) functions, such a function requires the use of signals and the reservation of some signal number. This volume of POSIX.1-2017 requires that *nanosleep*() be non-intrusive of the signals function.

The *nanosleep*() function shall return a value of 0 on success and −1 on failure or if interrupted. This latter case is different from *sleep*(). This was done because the remaining time is returned via an argument structure pointer, *rmtp*, instead of as the return value.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_nanosleep( ), *sleep*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nearbyint, nearbyintf, nearbyintl — floating-point rounding functions

SYNOPSIS

```
#include <math.h>

double nearbyint(double x);
float nearbyintf(float x);
long double nearbyintl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to an integer value in floating-point format, using the current rounding direction and without raising the inexact floating-point exception.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value. The result shall have the same sign as *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, ±0 shall be returned.

If *x* is ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

newlocale — create or modify a locale object

SYNOPSIS

```
#include <locale.h>

locale_t newlocale(int category_mask, const char *locale,
    locale_t base);
```

DESCRIPTION

The *newlocale*() function shall create a new locale object or modify an existing one. If the *base* argument is (**locale_t**)0, a new locale object shall be created. It is unspecified whether the locale object pointed to by *base* shall be modified, or freed and a new locale object created.

The *category_mask* argument specifies the locale categories to be set or modified. Values for *category_mask* shall be constructed by a bitwise-inclusive OR of the symbolic constants *LC_CTYPE_MASK*, *LC_NUMERIC_MASK*, *LC_TIME_MASK*, *LC_COLLATE_MASK*, *LC_MONETARY_MASK*, and *LC_MESSAGES_MASK*, or any of the implementation-defined mask values defined in *<locale.h>*.

For each category with the corresponding bit set in *category_mask* the data from the locale named by *locale* shall be used. In the case of modifying an existing locale object, the data from the locale named by *locale* shall replace the existing data within the locale object. If a completely new locale object is created, the data for all sections not requested by *category_mask* shall be taken from the default locale.

The following preset values of *locale* are defined for all settings of *category_mask*:

"POSIX"	Specifies the minimal environment for C-language translation called the POSIX locale.
"C"	Equivalent to **"POSIX"**.
" "	Specifies an implementation-defined native environment. This corresponds to the value of the associated environment variables, *LC_** and *LANG*; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale* and *Chapter 8*, *Environment Variables*.

If the *base* argument is not (**locale_t**)0 and the *newlocale*() function call succeeds, the contents of *base* are unspecified. Applications shall ensure that they stop using *base* as a locale object before calling *newlocale*(). If the function call fails and the *base* argument is not (**locale_t**)0, the contents of *base* shall remain valid and unchanged.

The behavior is undefined if the *base* argument is the special locale object LC_GLOBAL_LOCALE, or is not a valid locale object handle and is not (**locale_t**)0.

RETURN VALUE

Upon successful completion, the *newlocale*() function shall return a handle which the caller may use on subsequent calls to *duplocale*(), *freelocale*(), and other functions taking a **locale_t** argument.

Upon failure, the *newlocale*() function shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *newlocale*() function shall fail if:

ENOMEM
: There is not enough memory available to create the locale object or load the locale data.

EINVAL
: The *category_mask* contains a bit that does not correspond to a valid category.

ENOENT
: For any of the categories in *category_mask*, the locale data is not available.

The *newlocale*() function may fail if:

EINVAL
The *locale* argument is not a valid string pointer.

The following sections are informative.

EXAMPLES

Constructing a Locale Object from Different Locales

The following example shows the construction of a locale where the *LC_CTYPE* category data comes from a locale *loc1* and the *LC_TIME* category data from a locale *loc2*:

```
#include <locale.h>
...
locale_t loc, new_loc;

/* Get the "loc1" data. */

loc = newlocale (LC_CTYPE_MASK, "loc1", (locale_t)0);
if (loc == (locale_t) 0)
    abort ();

/* Get the "loc2" data. */

new_loc = newlocale (LC_TIME_MASK, "loc2", loc);
if (new_loc != (locale_t) 0)
    /* We don t abort if this fails. In this case this
       simply used to unchanged locale object. */
    loc = new_loc;

...
```

Freeing up a Locale Object

The following example shows a code fragment to free a locale object created by *newlocale*():

```
#include <locale.h>
...

/* Every locale object allocated with newlocale() should be
 * freed using freelocale():
 */

locale_t loc;

/* Get the locale. */

loc = newlocale (LC_CTYPE_MASK | LC_TIME_MASK, "locname", (locale_t)0);

/* ... Use the locale object ... */
...

/* Free the locale object resources. */
freelocale (loc);
```

APPLICATION USAGE

Handles for locale objects created by the *newlocale*() function should either be released by a corresponding call to *freelocale*(), or be used as a base locale to another *newlocale*() call.

The special locale object LC_GLOBAL_LOCALE must not be passed for the *base* argument, even when returned by the *uselocale*() function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *freelocale*( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, *Chapter 8*, *Environment Variables*, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nextafter, nextafterf, nextafterl, nexttoward, nexttowardf, nexttowardl — next representable floating-point number

SYNOPSIS

```
#include <math.h>

double nextafter(double x, double y);
float nextafterf(float x, float y);
long double nextafterl(long double x, long double y);
double nexttoward(double x, long double y);
float nexttowardf(float x, long double y);
long double nexttowardl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *nextafter*(), *nextafterf*(), and *nextafterl*() functions shall compute the next representable floating-point value following *x* in the direction of *y*. Thus, if *y* is less than *x*, *nextafter*() shall return the largest representable floating-point number less than *x*. The *nextafter*(), *nextafterf*(), and *nextafterl*() functions shall return *y* if *x* equals *y*.

The *nexttoward*(), *nexttowardf*(), and *nexttowardl*() functions shall be equivalent to the corresponding *nextafter*() functions, except that the second parameter shall have type **long double** and the functions shall return *y* converted to the type of the function if *x* equals *y*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the next representable floating-point value following *x* in the direction of *y*.

If *x*==*y*, *y* (of the type *x*) shall be returned.

If *x* is finite and the correct function value would overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the return type of the function.

If *x* or *y* is NaN, a NaN shall be returned.

If *x*!=*y* and the correct function value is subnormal, zero, or underflows, a range error shall occur, and
the correct function value (if representable) or
0.0 shall be returned.

ERRORS

These functions shall fail if:

Range Error The correct value overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

Range Error The correct value is subnormal or underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

When *<tgmath.h>* is included, note that the return type of *nextafter*() depends on the generic typing deduced from both arguments, while the return type of *nexttoward*() depends only on the generic typing of the first argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**, **<tgmath.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nftw — walk a file tree

SYNOPSIS

```
#include <ftw.h>

int nftw(const char *path, int (*fn)(const char *,
   const struct stat *, int, struct FTW *), int fd_limit, int flags);
```

DESCRIPTION

The *nftw*() function shall recursively descend the directory hierarchy rooted in *path*. The *nftw*() function has a similar effect to *ftw*() except that it takes an additional argument *flags*, which is a bitwise-inclusive OR of zero or more of the following flags:

FTW_CHDIR
: If set, *nftw*() shall change the current working directory to each directory as it reports files in that directory. If clear, *nftw*() shall not change the current working directory.

FTW_DEPTH
: If set, *nftw*() shall report all files in a directory before reporting the directory itself. If clear, *nftw*() shall report any directory before reporting the files in that directory.

FTW_MOUNT
: If set, *nftw*() shall only report files in the same file system as *path*. If clear, *nftw*() shall report all files encountered during the walk.

FTW_PHYS
: If set, *nftw*() shall perform a physical walk and shall not follow symbolic links.

If FTW_PHYS is clear and FTW_DEPTH is set, *nftw*() shall follow links instead of reporting them, but shall not report any directory that would be a descendant of itself. If FTW_PHYS is clear and FTW_DEPTH is clear, *nftw*() shall follow links instead of reporting them, but shall not report the contents of any directory that would be a descendant of itself.

At each file it encounters, *nftw*() shall call the user-supplied function *fn* with four arguments:

* The first argument is the pathname of the object.
* The second argument is a pointer to the **stat** buffer containing information on the object, filled in as if *fstatat*(), *stat*(), or *lstat*() had been called to retrieve the information.
* The third argument is an integer giving additional information. Its value is one of the following:

  FTW_D
  : The object is a directory.

  FTW_DNR
  : The object is a directory that cannot be read. The *fn* function shall not be called for any of its descendants.

  FTW_DP
  : The object is a directory and subdirectories have been visited. (This condition shall only occur if the FTW_DEPTH flag is included in *flags*.)

  FTW_F
  : The object is a non-directory file.

  FTW_NS
  : The *stat*() function failed on the object because of lack of appropriate permission. The **stat** buffer passed to *fn* is undefined. Failure of *stat*() for any other reason is considered an error and *nftw*() shall return −1.

  FTW_SL
  : The object is a symbolic link. (This condition shall only occur if the FTW_PHYS flag is included in *flags*.)

FTW_SLN The object is a symbolic link that does not name an existing file. (This condition shall only occur if the FTW_PHYS flag is not included in *flags*.)

* The fourth argument is a pointer to an **FTW** structure. The value of **base** is the offset of the object's filename in the pathname passed as the first argument to *fn*. The value of **level** indicates depth relative to the root of the walk, where the root level is 0.

The results are unspecified if the application-supplied *fn* function does not preserve the current working directory.

The argument *fd_limit* sets the maximum number of file descriptors that shall be used by *nftw*() while traversing the file tree. At most one file descriptor shall be used for each directory level.

The *nftw*() function need not be thread-safe.

RETURN VALUE

The *nftw*() function shall continue until the first of the following conditions occurs:

* An invocation of *fn* shall return a non-zero value, in which case *nftw*() shall return that value.
* The *nftw*() function detects an error other than **[EACCES]** (see FTW_DNR and FTW_NS above), in which case *nftw*() shall return −1 and set *errno* to indicate the error.
* The tree is exhausted, in which case *nftw*() shall return 0.

ERRORS

The *nftw*() function shall fail if:

EACCES
Search permission is denied for any component of *path* or read permission is denied for *path*, or *fn* returns −1 and does not reset *errno*.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of *path* names an existing file that is neither a directory nor a symbolic link to a directory.

EOVERFLOW
A field in the **stat** structure cannot be represented correctly in the current programming environment for one or more files found in the file hierarchy.

The *nftw*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENFILE
Too many files are currently open in the system.

In addition, *errno* may be set if the function pointed to by *fn* causes *errno* to be set.

The following sections are informative.

EXAMPLES

The following program traverses the directory tree under the path named in its first command-line argument, or under the current directory if no argument is supplied. It displays various information about each file. The second command-line argument can be used to specify characters that control the value assigned to the *flags* argument when calling *nftw*().

```
#include <ftw.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>

static int
display_info(const char *fpath, const struct stat *sb,
    int tflag, struct FTW *ftwbuf)
{
    printf("%-3s %2d %7jd   %-40s %d %s\n",
        (tflag == FTW_D) ?   "d"   : (tflag == FTW_DNR) ? "dnr" :
        (tflag == FTW_DP) ?  "dp"  : (tflag == FTW_F) ?
            (S_ISBLK(sb->st_mode) ? "f b" :
             S_ISCHR(sb->st_mode) ? "f c" :
             S_ISFIFO(sb->st_mode) ? "f p" :
             S_ISREG(sb->st_mode) ? "f r" :
             S_ISSOCK(sb->st_mode) ? "f s" : "f ?") :
        (tflag == FTW_NS) ?  "ns"  : (tflag == FTW_SL) ?  "sl" :
        (tflag == FTW_SLN) ? "sln" : "?",
        ftwbuf->level, (intmax_t) sb->st_size,
        fpath, ftwbuf->base, fpath + ftwbuf->base);
    return 0;           /* To tell nftw() to continue */
}

int
main(int argc, char *argv[])
{
    int flags = 0;

    if (argc > 2 && strchr(argv[2], 'd') != NULL)
        flags |= FTW_DEPTH;
    if (argc > 2 && strchr(argv[2], 'p') != NULL)
        flags |= FTW_PHYS;

    if (nftw((argc < 2) ? "." : argv[1], display_info, 20, flags) == -1)
    {
        perror("nftw");
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

APPLICATION USAGE

The *nftw*() function may allocate dynamic storage during its operation. If *nftw*() is forcibly terminated, such as by *longjmp*() or *siglongjmp*() being executed by the function pointed to by *fn* or an interrupt routine, *nftw*() does not have a chance to free that storage, so it remains permanently allocated. A safe way to handle interrupts is to store the fact that an interrupt has occurred, and arrange to have the function pointed to by *fn* return a non-zero value at its next invocation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir( ), *fstatat*( ), *readdir*( )

The Base Definitions volume of POSIX.1-2017, **<ftw.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nice — change the nice value of a process

SYNOPSIS

```
#include <unistd.h>

int nice(int incr);
```

DESCRIPTION

The *nice*() function shall add the value of *incr* to the nice value of the calling process. A nice value of a process is a non-negative number for which a more positive value shall result in less favorable scheduling.

A maximum nice value of 2*{NZERO}−1 and a minimum nice value of 0 shall be imposed by the system. Requests for values above or below these limits shall result in the nice value being set to the corresponding limit. Only a process with appropriate privileges can lower the nice value.

Calling the *nice*() function has no effect on the priority of processes or threads with policy SCHED_FIFO or SCHED_RR. The effect on processes or threads with other scheduling policies is implementation-defined.

The nice value set with *nice*() shall be applied to the process. If the process is multi-threaded, the nice value shall affect all system scope threads in the process.

As −1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *nice*(), and if it returns −1, check to see whether *errno* is non-zero.

RETURN VALUE

Upon successful completion, *nice*() shall return the new nice value −{NZERO}. Otherwise, −1 shall be returned, the nice value of the process shall not be changed, and *errno* shall be set to indicate the error.

ERRORS

The *nice*() function shall fail if:

EPERM
The *incr* argument is negative and the calling process does not have appropriate privileges.

The following sections are informative.

EXAMPLES

Changing the Nice Value

The following example adds the value of the *incr* argument, −20, to the nice value of the calling process.

```
#include <unistd.h>
...
int incr = -20;
int ret;

ret = nice(incr);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpriority*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nl_langinfo, nl_langinfo_l — language information

SYNOPSIS

```
#include <langinfo.h>

char *nl_langinfo(nl_item item);
char *nl_langinfo_l(nl_item item, locale_t locale);
```

DESCRIPTION

The *nl_langinfo*() and *nl_langinfo_l*() functions shall return a pointer to a string containing information relevant to the particular language or cultural area defined in the current locale, or in the locale represented by *locale*, respectively (see *<langinfo.h>*). The manifest constant names and values of *item* are defined in *<langinfo.h>*. For example:

```
    nl_langinfo(ABDAY_1)
```

would return a pointer to the string **"Dom"** if the identified language was Portuguese, and **"Sun"** if the identified language was English.

```
    nl_langinfo_l(ABDAY_1, loc)
```

would return a pointer to the string **"Dom"** if the identified language of the locale represented by *loc* was Portuguese, and **"Sun"** if the identified language of the locale represented by *loc* was English.

The *nl_langinfo*() function need not be thread-safe.

The behavior is undefined if the *locale* argument to *nl_langinfo_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

In a locale where *langinfo* data is not defined, these functions shall return a pointer to the corresponding string in the POSIX locale. In all locales, these functions shall return a pointer to an empty string if *item* contains an invalid setting.

The application shall not modify the string returned. The pointer returned by *nl_langinfo*() might be invalidated or the string content might be overwritten by a subsequent call to *nl_langinfo*() in any thread or to *nl_langinfo_l*() in the same thread or the initial thread, by subsequent calls to *setlocale*() with a category corresponding to the category of *item* (see *<langinfo.h>*) or the category LC_ALL, or by subsequent calls to *uselocale*() which change the category corresponding to the category of *item*. The pointer returned by *nl_langinfo_l*() might be invalidated or the string content might be overwritten by a subsequent call to *nl_langinfo_l*() in the same thread or to *nl_langinfo*() in any thread, or by subsequent calls to *freelocale*() or *newlocale*() which free or modify the locale object that was passed to *nl_langinfo_l*(). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Getting Date and Time Formatting Information

The following example returns a pointer to a string containing date and time formatting information, as defined in the *LC_TIME* category of the current locale.

```
#include <time.h>
#include <langinfo.h>
...
strftime(datestring, sizeof(datestring), nl_langinfo(D_T_FMT), tm);
...
```

APPLICATION USAGE

The array pointed to by the return value should not be modified by the program, but may be modified by further calls to these functions.

RATIONALE

The possible interactions between internal data used by *nl_langinfo*() and *nl_langinfo_l*() are complicated by the fact that *nl_langinfo_l*() must be thread-safe but *nl_langinfo*() need not be. The various implementation choices are:

1. *nl_langinfo_l*() and *nl_langinfo*() use separate buffers, or at least one of them does not use an internal string buffer. In this case there are no interactions.
2. *nl_langinfo_l*() and *nl_langinfo*() share an internal per-thread buffer. There can be interactions, but only in the same thread.
3. *nl_langinfo_l*() uses an internal per-thread buffer, and *nl_langinfo*() uses (in all threads) the same buffer that *nl_langinfo_l*() uses in the initial thread. There can be interactions, but only when *nl_langinfo_l*() is called in the initial thread.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<langinfo.h>**, **<locale.h>**, **<nl_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

nrand48 — generate uniformly distributed pseudo-random non-negative long integers

SYNOPSIS

#include <stdlib.h>

long nrand48(unsigned short *xsubi*[3]);

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ntohl, ntohs — convert values between host and network byte order

SYNOPSIS

```
#include <arpa/inet.h>

uint32_t ntohl(uint32_t netlong);
uint16_t ntohs(uint16_t netshort);
```

DESCRIPTION

Refer to *htonl*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

open, openat — open file

SYNOPSIS

```
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *path, int oflag, ...);
int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

The *open*() function shall establish the connection between a file and a file descriptor. It shall create an open file description that refers to a file and a file descriptor that refers to that open file description. The file descriptor is used by other I/O functions to refer to that file. The *path* argument points to a pathname naming the file.

The *open*() function shall return a file descriptor for the named file, allocated as described in *Section 2.14, File Descriptor Allocation*. The open file description is new, and therefore the file descriptor shall not share it with any other process in the system. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared unless the O_CLOEXEC flag is set in *oflag*.

The file offset used to mark the current position within the file shall be set to the beginning of the file.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*. Applications shall specify exactly one of the first five values (file access modes) below in the value of *oflag*:

O_EXEC	Open for execute only (non-directory files). The result is unspecified if this flag is applied to a directory.
O_RDONLY	Open for reading only.
O_RDWR	Open for reading and writing. The result is undefined if this flag is applied to a FIFO.
O_SEARCH	Open directory for search only. The result is unspecified if this flag is applied to a non-directory file.
O_WRONLY	Open for writing only.

Any combination of the following may be used:

O_APPEND	If set, the file offset shall be set to the end of the file prior to each write.
O_CLOEXEC	If set, the FD_CLOEXEC flag for the new file descriptor shall be set.
O_CREAT	If the file exists, this flag has no effect except as noted under O_EXCL below. Otherwise, if O_DIRECTORY is not set the file shall be created as a regular file; the user ID of the file shall be set to the effective user ID of the process; the group ID of the file shall be set to the group ID of the file's parent directory or to the effective group ID of the process; and the access permission bits (see *<sys/stat.h>*) of the file mode shall be set to the value of the argument following the *oflag* argument taken as type **mode_t** modified as follows: a bitwise AND is performed on the file-mode bits and the corresponding bits in the complement of the process' file mode creation mask. Thus, all bits in the file mode whose corresponding bit in the file mode creation mask is set are cleared. When bits other than the file permission bits are set, the effect is unspecified. The argument following the *oflag* argument does not affect whether the file is open for reading, writing, or for both. Implementations shall provide a way to initialize the file's group ID to the group ID of the parent

directory. Implementations may, but need not, provide an implementation-defined way to initialize the file's group ID to the effective group ID of the calling process.

O_DIRECTORY
: If *path* resolves to a non-directory file, fail and set *errno* to **[ENOTDIR]**.

O_DSYNC
: Write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

O_EXCL
: If O_CREAT and O_EXCL are set, *open*() shall fail if the file exists. The check for the existence of the file and the creation of the file if it does not exist shall be atomic with respect to other threads executing *open*() naming the same filename in the same directory with O_EXCL and O_CREAT set. If O_EXCL and O_CREAT are set, and *path* names a symbolic link, *open*() shall fail and set *errno* to **[EEXIST]**, regardless of the contents of the symbolic link. If O_EXCL is set and O_CREAT is not set, the result is undefined.

O_NOCTTY
: If set and *path* identifies a terminal device, *open*() shall not cause the terminal device to become the controlling terminal for the process. If *path* does not identify a terminal device, O_NOCTTY shall be ignored.

O_NOFOLLOW
: If *path* names a symbolic link, fail and set *errno* to **[ELOOP]**.

O_NONBLOCK
: When opening a FIFO with O_RDONLY or O_WRONLY set:

* If O_NONBLOCK is set, an *open*() for reading-only shall return without delay. An *open*() for writing-only shall return an error if no process currently has the file open for reading.
* If O_NONBLOCK is clear, an *open*() for reading-only shall block the calling thread until a thread opens the file for writing. An *open*() for writing-only shall block the calling thread until a thread opens the file for reading.

When opening a block special or character special file that supports non-blocking opens:

* If O_NONBLOCK is set, the *open*() function shall return without blocking for the device to be ready or available. Subsequent behavior of the device is device-specific.
* If O_NONBLOCK is clear, the *open*() function shall block the calling thread until the device is ready or available before returning.

Otherwise, the O_NONBLOCK flag shall not cause an error, but it is unspecified whether the file status flags will include the O_NONBLOCK flag.

O_RSYNC
: Read I/O operations on the file descriptor shall complete at the same level of integrity as specified by the O_DSYNC and O_SYNC flags. If both O_DSYNC and O_RSYNC are set in *oflag*, all I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If both O_SYNC and O_RSYNC are set in flags, all I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

O_SYNC
: Write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

The O_SYNC flag shall be supported for regular files, even if the Synchronized Input and Output option is not supported.

O_TRUNC
: If the file exists and is a regular file, and the file is successfully opened O_RDWR or O_WRONLY, its length shall be truncated to 0, and the mode and owner shall be unchanged. It shall have no effect on FIFO special files or terminal device files. Its effect on other file types is implementation-defined. The result of using O_TRUNC without either O_RDWR or O_WRONLY is undefined.

O_TTY_INIT If *path* identifies a terminal device other than a pseudo-terminal, the device is not already open in any process, and either O_TTY_INIT is set in *oflag* or O_TTY_INIT has the value zero, *open*() shall set any non-standard **termios** structure terminal parameters to a state that provides conforming behavior; see the Base Definitions volume of POSIX.1-2017, *Section 11.2*, *Parameters that Can be Set*. It is unspecified whether O_TTY_INIT has any effect if the device is already open in any process. If *path* identifies the slave side of a pseudo-terminal that is not already open in any process, *open*() shall set any non-standard **termios** structure terminal parameters to a state that provides conforming behavior, regardless of whether O_TTY_INIT is set. If *path* does not identify a terminal device, O_TTY_INIT shall be ignored.

If O_CREAT and O_DIRECTORY are set and the requested access mode is neither O_WRONLY nor O_RDWR, the result is unspecified.

If O_CREAT is set and the file did not previously exist, upon successful completion, *open*() shall mark for update the last data access, last data modification, and last file status change timestamps of the file and the last data modification and last file status change timestamps of the parent directory.

If O_TRUNC is set and the file did previously exist, upon successful completion, *open*() shall mark for update the last data modification and last file status change timestamps of the file.

If both the O_SYNC and O_DSYNC flags are set, the effect is as if only the O_SYNC flag was set.

If *path* refers to a STREAMS file, *oflag* may be constructed from O_NONBLOCK OR'ed with either O_RDONLY, O_WRONLY, or O_RDWR. Other flag values are not applicable to STREAMS devices and shall have no effect on them. The value O_NONBLOCK affects the operation of STREAMS drivers and certain functions applied to file descriptors associated with STREAMS files. For STREAMS drivers, the implementation of O_NONBLOCK is device-specific.

The application shall ensure that it specifies the O_TTY_INIT flag on the first open of a terminal device since system boot or since the device was closed by the process that last had it open. The application need not specify the O_TTY_INIT flag when opening pseudo-terminals. If *path* names the master side of a pseudo-terminal device, then it is unspecified whether *open*() locks the slave side so that it cannot be opened. Conforming applications shall call *unlockpt*() before opening the slave side.

The largest value that can be represented correctly in an object of type **off_t** shall be established as the offset maximum in the open file description.

The *openat*() function shall be equivalent to the *open*() function except in the case where *path* specifies a relative path. In this case the file to be opened is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

The *oflag* parameter and the optional fourth parameter correspond exactly to the parameters of *open*().

If *openat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *open*().

RETURN VALUE

Upon successful completion, these functions shall open the file and return a non-negative integer representing the file descriptor. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, no files shall be created or modified.

ERRORS

These functions shall fail if:

EACCES

Search permission is denied on a component of the path prefix, or the file exists and the permissions specified by *oflag* are denied, or the file does not exist and write permission is denied for the parent directory of the file to be created, or O_TRUNC is specified and write permission is denied.

EEXIST
O_CREAT and O_EXCL are set, and the named file exists.

EINTR
A signal was caught during *open*().

EINVAL
The implementation does not support synchronized I/O for this file.

EIO The *path* argument names a STREAMS file and a hangup or error occurred during the *open*().

EISDIR
The named file is a directory and *oflag* includes O_WRONLY or O_RDWR, or includes O_CREAT without O_DIRECTORY.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument, or O_NOFOLLOW was specified and the *path* argument names a symbolic link.

EMFILE
All file descriptors available to the process are currently open.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENFILE
The maximum allowable number of files is currently open in the system.

ENOENT
O_CREAT is not set and a component of *path* does not name an existing file, or O_CREAT is set and a component of the path prefix of *path* does not name an existing file, or *path* points to an empty string.

ENOENT or **ENOTDIR**
O_CREAT is set, and the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSR
The *path* argument names a STREAMS-based file and the system is unable to allocate a STREAM.

ENOSPC
The directory or file system that would contain the new file cannot be expanded, the file does not exist, and O_CREAT is specified.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or O_CREAT and O_EXCL are not specified, the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters, and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or O_DIRECTORY was specified and the *path* argument resolves to a non-directory file.

ENXIO
O_NONBLOCK is set, the named file is a FIFO, O_WRONLY is set, and no process has the file open for reading.

ENXIO
The named file is a character special or block special file, and the device associated with this special file does not exist.

EOVERFLOW
The named file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

EROFS
The named file resides on a read-only file system and either O_WRONLY, O_RDWR, O_CREAT (if the file does not exist), or O_TRUNC is set in the *oflag* argument.

The *openat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

EAGAIN
The *path* argument names the slave side of a pseudo-terminal device that is locked.

EINVAL
The value of the *oflag* argument is not valid.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENOMEM
The *path* argument names a STREAMS file and the system is unable to allocate resources.

EOPNOTSUPP
The *path* argument names a socket.

ETXTBSY
The file is a pure procedure (shared text) file that is being executed and *oflag* is O_WRONLY or O_RDWR.

The following sections are informative.

EXAMPLES

Opening a File for Writing by the Owner

The following example opens the file **/tmp/file**, either by creating it (if it does not already exist), or by truncating its length to 0 (if it does exist). In the former case, if the call creates a new file, the access permission bits in the file mode of the file are set to permit reading and writing by the owner, and to permit reading only by group members and others.

If the call to *open*() is successful, the file is opened for writing.

```
#include <fcntl.h>
...
int fd;
mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH;
char *pathname = "/tmp/file";
...
fd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC, mode);
```

```
...
```

Opening a File Using an Existence Check

The following example uses the *open*() function to try to create the **LOCKFILE** file and open it for writing. Since the *open*() function specifies the O_EXCL flag, the call fails if the file already exists. In that case, the program assumes that someone else is updating the password file and exits.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd; /* Integer for file descriptor returned by open() call. */
...
if ((pfd = open(LOCKFILE, O_WRONLY | O_CREAT | O_EXCL,
   S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
   fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
   exit(1);
}
...
```

Opening a File for Writing

The following example opens a file for writing, creating the file if it does not already exist. If the file does exist, the system truncates the file to zero bytes.

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

#define LOCKFILE "/etc/ptmp"
...
int pfd;
char pathname[PATH_MAX+1];
...
if ((pfd = open(pathname, O_WRONLY | O_CREAT | O_TRUNC,
   S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)) == -1)
{
   perror("Cannot open output file\n"); exit(1);
}
...
```

APPLICATION USAGE

POSIX.1-2008 does not require that terminal parameters be automatically set to any state on first open, nor that they be reset after the last close. It is possible for a non-conforming application to leave a terminal device in a state where the next process to use that device finds it in a non-conforming state, but has no way of determining this. To ensure that the device is set to a conforming initial state, applications which perform a first open of a terminal (other than a pseudo-terminal) should do so using the O_TTY_INIT flag to set the parameters associated with the terminal to a conforming state.

Except as specified in this volume of POSIX.1-2017, the flags allowed in *oflag* are not mutually-exclusive and any number of them may be used simultaneously. Not all combinations of flags make sense. For example, using O_SEARCH | O_CREAT will successfully open a pre-existing directory for searching, but if there is no existing file by that name, then it is unspecified whether a regular file will be created. Likewise, if a non-directory file descriptor is successfully returned, it is unspecified whether that descriptor will have execute permissions as if by O_EXEC (note that it is unspecified whether O_EXEC and O_SEARCH have the same value).

RATIONALE

Some implementations permit opening FIFOs with O_RDWR. Since FIFOs could be implemented in other ways, and since two file descriptors can be used to the same effect, this possibility is left as undefined.

See *getgroups*( ) about the group of a newly created file.

The use of *open*() to create a regular file is preferable to the use of *creat*(), because the latter is redundant and included only for historical reasons.

The use of the O_TRUNC flag on FIFOs and directories (pipes cannot be *open*()-ed) must be permissible without unexpected side-effects (for example, *creat*() on a FIFO must not remove data). Since terminal special files might have type-ahead data stored in the buffer, O_TRUNC should not affect their content, particularly if a program that normally opens a regular file should open the current controlling terminal instead. Other file types, particularly implementation-defined ones, are left implementation-defined.

POSIX.1-2008 permits **[EACCES]** to be returned for conditions other than those explicitly listed.

The O_NOCTTY flag was added to allow applications to avoid unintentionally acquiring a controlling terminal as a side-effect of opening a terminal file. This volume of POSIX.1-2017 does not specify how a controlling terminal is acquired, but it allows an implementation to provide this on *open*() if the O_NOCTTY flag is not set and other conditions specified in the Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface* are met.

In historical implementations the value of O_RDONLY is zero. Because of that, it is not possible to detect the presence of O_RDONLY and another option. Future implementations should encode O_RDONLY and O_WRONLY as bit flags so that:

```
O_RDONLY | O_WRONLY == O_RDWR
```

O_EXEC and O_SEARCH are specified as two of the five file access modes. Since O_EXEC does not apply to directories, and O_SEARCH only applies to directories, their values need not be distinct. Since O_RDONLY has historically had the value zero, implementations are not able to distinguish between O_SEARCH and O_SEARCH | O_RDONLY, and similarly for O_EXEC.

In general, the *open*() function follows the symbolic link if *path* names a symbolic link. However, the *open*() function, when called with O_CREAT and O_EXCL, is required to fail with **[EEXIST]** if *path* names an existing symbolic link, even if the symbolic link refers to a nonexistent file. This behavior is required so that privileged applications can create a new file in a known location without the possibility that a symbolic link might cause the file to be created in a different location.

For example, a privileged application that must create a file with a predictable name in a user-writable directory, such as the user's home directory, could be compromised if the user creates a symbolic link with that name that refers to a nonexistent file in a system directory. If the user can influence the contents of a file, the user could compromise the system by creating a new system configuration or spool file that would then be interpreted by the system. The test for a symbolic link which refers to a nonexisting file must be atomic with the creation of a new file.

In addition, the *open*() function refuses to open non-directories if the O_DIRECTORY flag is set. This avoids race conditions whereby a user might compromise the system by substituting a hard link to a sensitive file (e.g., a device or a FIFO) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

In addition, the *open*() function does not follow symbolic links if the O_NOFOLLOW flag is set. This avoids race conditions whereby a user might compromise the system by substituting a symbolic link to a sensitive file (e.g., a device) while a privileged application is running, where opening a file even for read access might have undesirable side-effects.

The POSIX.1-1990 standard required that the group ID of a newly created file be set to the group ID of its parent directory or to the effective group ID of the creating process. FIPS 151-2 required that implementations provide a way to have the group ID be set to the group ID of the containing directory, but did not prohibit implementations also supporting a way to set the group ID to the effective group ID of the creating process. Conforming applications should not assume which group ID will be used. If it matters, an application can use *chown*() to set the group ID after the file is created, or determine under what conditions the implementation will set the desired group ID.

The purpose of the *openat*() function is to enable opening files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *open*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *openat*() function it can be guaranteed that the opened file is located relative to the desired directory. Some implementations use the *openat*() function for other purposes as well. In some cases, if the *oflag* parameter has the O_XATTR bit set, the returned file descriptor provides access to extended attributes. This functionality is not standardized here.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *close*( ), *creat*( ), *dirfd*( ), *dup*( ), *exec*, *fcntl*( ), *fdopendir*( ), *link*( ), *lseek*( ), *mkdtemp*( ), *mknod*( ), *read*( ), *symlink*( ), *umask*( ), *unlockpt*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<fcntl.h>**, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

open_memstream, open_wmemstream — open a dynamic memory buffer stream

SYNOPSIS

#include <stdio.h>

FILE *open_memstream(char ***bufp*, size_t **sizep*);

#include <wchar.h>

FILE *open_wmemstream(wchar_t ***bufp*, size_t **sizep*);

DESCRIPTION

The *open_memstream*() and *open_wmemstream*() functions shall create an I/O stream associated with a dynamically allocated memory buffer. The stream shall be opened for writing and shall be seekable.

The stream associated with a call to *open_memstream*() shall be byte-oriented.

The stream associated with a call to *open_wmemstream*() shall be wide-oriented.

The stream shall maintain a current position in the allocated buffer and a current buffer length. The position shall be initially set to zero (the start of the buffer). Each write to the stream shall start at the current position and move this position by the number of successfully written bytes for *open_memstream*() or the number of successfully written wide characters for *open_wmemstream*(). The length shall be initially set to zero. If a write moves the position to a value larger than the current length, the current length shall be set to this position. In this case a null character for *open_memstream*() or a null wide character for *open_wmemstream*() shall be appended to the current buffer. For both functions the terminating null is not included in the calculation of the buffer length.

After a successful *fflush*() or *fclose*(), the pointer referenced by *bufp* shall contain the address of the buffer, and the variable pointed to by *sizep* shall contain the smaller of the current buffer length and the number of bytes for *open_memstream*(), or the number of wide characters for *open_wmemstream*(), between the beginning of the buffer and the current file position indicator.

After a successful *fflush*() the pointer referenced by *bufp* and the variable referenced by *sizep* remain valid only until the next write operation on the stream or a call to *fclose*().

After a successful *fclose*(), the pointer referenced by *bufp* can be passed to *free*().

RETURN VALUE

Upon successful completion, these functions shall return a pointer to the object controlling the stream. Otherwise, a null pointer shall be returned, and *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

These functions may fail if:

EINVAL
bufp or *sizep* are NULL.

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

ENOMEM
Memory for the stream or the buffer could not be allocated.

The following sections are informative.

EXAMPLES

```
#include <stdio.h>
#include <stdlib.h>

int
main (void)
{
   FILE *stream;
   char *buf;
   size_t len;
   off_t eob;

   stream = open_memstream (&buf, &len);
   if (stream == NULL)
      /* handle error */ ;
   fprintf (stream, "hello my world");
   fflush (stream);
   printf ("buf=%s, len=%zu\n", buf, len);
   eob = ftello(stream);
   fseeko (stream, 0, SEEK_SET);
   fprintf (stream, "good-bye");
   fseeko (stream, eob, SEEK_SET);
   fclose (stream);
   printf ("buf=%s, len=%zu\n", buf, len);
   free (buf);
   return 0;
}
```

This program produces the following output:

```
buf=hello my world, len=14
buf=good-bye world, len=14
```

APPLICATION USAGE

The buffer created by these functions should be freed by the application after closing the stream, by means of a call to *free*().

RATIONALE

These functions are similar to *fmemopen*() except that the memory is always allocated dynamically by the function, and the stream is opened only for output.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *fdopen*( ), *fflush*( ), *fmemopen*( ), *fopen*( ), *free*( ), *freopen*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

openat — open file relative to directory file descriptor

SYNOPSIS

```
#include <fcntl.h>

int openat(int fd, const char *path, int oflag, ...);
```

DESCRIPTION

Refer to *open*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

opendir — open directory associated with file descriptor

SYNOPSIS

```
#include <dirent.h>

DIR *opendir(const char *dirname);
```

DESCRIPTION

Refer to *fdopendir*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

openlog — open a connection to the logging facility

SYNOPSIS

#include <syslog.h>

void openlog(const char **ident*, int *logopt*, int *facility*);

DESCRIPTION

Refer to *closelog*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

optarg, opterr, optind, optopt — options parsing variables

SYNOPSIS

```
#include <unistd.h>

extern char *optarg;
extern int opterr, optind, optopt;
```

DESCRIPTION

Refer to *getopt*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pathconf — get configurable pathname variables

SYNOPSIS

#include <unistd.h>

long pathconf(const char **path*, int *name*);

DESCRIPTION

Refer to *fpathconf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pause — suspend the thread until a signal is received

SYNOPSIS

```
#include <unistd.h>

int pause(void);
```

DESCRIPTION

The *pause*() function shall suspend the calling thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, *pause*() shall not return.

If the action is to execute a signal-catching function, *pause*() shall return after the signal-catching function returns.

RETURN VALUE

Since *pause*() suspends thread execution indefinitely unless interrupted by a signal, there is no successful completion return value. A value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *pause*() function shall fail if:

EINTR

A signal is caught by the calling process and control is returned from the signal-catching function.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Many common uses of *pause*() have timing windows. The scenario involves checking a condition related to a signal and, if the signal has not occurred, calling *pause*(). When the signal occurs between the check and the call to *pause*(), the process often blocks indefinitely. The *sigprocmask*() and *sigsuspend*() functions can be used to avoid this type of problem.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sigsuspend( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pclose — close a pipe stream to or from a process

SYNOPSIS

```
#include <stdio.h>

int pclose(FILE *stream);
```

DESCRIPTION

The *pclose*() function shall close a stream that was opened by *popen*(), wait for the command to terminate, and return the termination status of the process that was running the command language interpreter. However, if a call caused the termination status to be unavailable to *pclose*(), then *pclose*() shall return −1 with *errno* set to **[ECHILD]** to report this situation. This can happen if the application calls one of the following functions:

* *wait*()
* *waitpid*() with a *pid* argument less than or equal to 0 or equal to the process ID of the command line interpreter
* Any other function not defined in this volume of POSIX.1-2017 that could do one of the above

In any case, *pclose*() shall not return before the child process created by *popen*() has terminated.

If the command language interpreter cannot be executed, the child termination status returned by *pclose*() shall be as if the command language interpreter terminated using *exit*(127) or *_exit*(127).

The *pclose*() function shall not affect the termination status of any child of the calling process other than the one created by *popen*() for the associated stream.

If the argument *stream* to *pclose*() is not a pointer to a stream created by *popen*(), the result of *pclose*() is undefined.

If a thread is canceled during execution of *pclose*(), the behavior is undefined.

RETURN VALUE

Upon successful return, *pclose*() shall return the termination status of the command language interpreter. Otherwise, *pclose*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *pclose*() function shall fail if:

ECHILD
The status of the child process could not be obtained, as described above.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There is a requirement that *pclose*() not return before the child process terminates. This is intended to disallow implementations that return **[EINTR]** if a signal is received while waiting. If *pclose*() returned before the child terminated, there would be no way for the application to discover which child used to be associated with the stream, and it could not do the cleanup itself.

If the stream pointed to by *stream* was not created by *popen*(), historical implementations of *pclose*() return −1 without setting *errno*. To avoid requiring *pclose*() to set *errno* in this case, POSIX.1-2008 makes the behavior unspecified. An application should not use *pclose*() to close any stream that was not created by *popen*().

Some historical implementations of *pclose*() either block or ignore the signals SIGINT, SIGQUIT, and SIGHUP while waiting for the child process to terminate. Since this behavior is not described for the *pclose*() function in POSIX.1-2008, such implementations are not conforming. Also, some historical implementations return **[EINTR]** if a signal is received, even though the child process has not terminated. Such implementations are also considered non-conforming.

Consider, for example, an application that uses:

```
popen("command", "r")
```

to start *command*, which is part of the same application. The parent writes a prompt to its standard output (presumably the terminal) and then reads from the *popen*()ed stream. The child reads the response from the user, does some transformation on the response (pathname expansion, perhaps) and writes the result to its standard output. The parent process reads the result from the pipe, does something with it, and prints another prompt. The cycle repeats. Assuming that both processes do appropriate buffer flushing, this would be expected to work.

To conform to POSIX.1-2008, *pclose*() must use *waitpid*(), or some similar function, instead of *wait*().

The code sample below illustrates how the *pclose*() function might be implemented on a system conforming to POSIX.1-2008.

```
int pclose(FILE *stream)
{
    int stat;
    pid_t pid;

    pid = <pid for process created for stream by popen()>
    (void) fclose(stream);
    while (waitpid(pid, &stat, 0) == -1) {
        if (errno != EINTR){
            stat = -1;
            break;
        }
    }
    return(stat);
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *popen*( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

perror — write error messages to standard error

SYNOPSIS

#include <stdio.h>

void perror(const char **s*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *perror*() function shall map the error number accessed through the symbol *errno* to a language-dependent error message, which shall be written to the standard error stream as follows:

* First (if *s* is not a null pointer and the character pointed to by *s* is not the null byte), the string pointed to by *s* followed by a <colon> and a <space>.
* Then an error message string followed by a <newline>.

The contents of the error message strings shall be the same as those returned by *strerror*() with argument *errno*.

The *perror*() function shall mark for update the last data modification and last file status change timestamps of the file associated with the standard error stream at some time between its successful completion and *exit*(), *abort*(), or the completion of *fflush*() or *fclose*() on *stderr*.

The *perror*() function shall not change the orientation of the standard error stream.

On error, *perror*() shall set the error indicator for the stream to which *stderr* points, and shall set *errno* to indicate the error.

Since no value is returned, an application wishing to check for error situations should call *clearerr*(*stderr*) before calling *perror*(), then if *ferror*(*stderr*) returns non-zero, the value of *errno* indicates which error occurred.

RETURN VALUE

The *perror*() function shall not return a value.

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

Printing an Error Message for a Function

The following example replaces *bufptr* with a buffer that is the necessary size. If an error occurs, the *perror*() function prints a message and the program exits.

```
#include <stdio.h>
#include <stdlib.h>
...
char *bufptr;
size_t szbuf;
...
if ((bufptr = malloc(szbuf)) == NULL) {
    perror("malloc"); exit(2);
    }
    ...
```

APPLICATION USAGE

Application writers may prefer to use alternative interfaces instead of *perror*(), such as *strerror_r*() in combination with *fprintf*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fputc*( ), *psiginfo*( ), *strerror*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pipe — create an interprocess channel

SYNOPSIS

#include <unistd.h>

int pipe(int *fildes*[2]);

DESCRIPTION

The *pipe*() function shall create a pipe and place two file descriptors, one each into the arguments *fildes*[0] and *fildes*[1], that refer to the open file descriptions for the read and write ends of the pipe. The file descriptors shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*. The O_NONBLOCK and FD_CLOEXEC flags shall be clear on both file descriptors. (The *fcntl*() function can be used to set both these flags.)

Data can be written to the file descriptor *fildes*[1] and read from the file descriptor *fildes*[0]. A read on the file descriptor *fildes*[0] shall access data written to the file descriptor *fildes*[1] on a first-in-first-out basis. It is unspecified whether *fildes*[0] is also open for writing and whether *fildes*[1] is also open for reading.

A process has the pipe open for reading (correspondingly writing) if it has a file descriptor open that refers to the read end, *fildes*[0] (write end, *fildes*[1]).

The pipe's user ID shall be set to the effective user ID of the calling process.

The pipe's group ID shall be set to the effective group ID of the calling process.

Upon successful completion, *pipe*() shall mark for update the last data access, last data modification, and last file status change timestamps of the pipe.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned and *errno* set to indicate the error, no file descriptors shall be allocated and the contents of *fildes* shall be left unmodified.

ERRORS

The *pipe*() function shall fail if:

EMFILE
All, or all but one, of the file descriptors available to the process are currently open.

ENFILE
The number of simultaneously open files in the system would exceed a system-imposed limit.

The following sections are informative.

EXAMPLES

Using a Pipe to Pass Data Between a Parent Process and a Child Process

The following example demonstrates the use of a pipe to transfer data between a parent process and a child process. Error handling is excluded, but otherwise this code demonstrates good practice when using pipes: after the *fork*() the two processes close the unused ends of the pipe before they commence transferring data.

```
#include <stdlib.h>
#include <unistd.h>
...

int fildes[2];
const int BSIZE = 100;
char buf[BSIZE];
ssize_t nbytes;
```

```
int status;

status = pipe(fildes);
if (status == -1 ) {
    /* an error occurred */
    ...
}

switch (fork()) {
case -1: /* Handle error */
    break;

case 0:  /* Child - reads from pipe */
    close(fildes[1]);                       /* Write end is unused */
    nbytes = read(fildes[0], buf, BSIZE);   /* Get data from pipe */
    /* At this point, a further read would see end-of-file ... */
    close(fildes[0]);                       /* Finished with pipe */
    exit(EXIT_SUCCESS);

default:  /* Parent - writes to pipe */
    close(fildes[0]);                       /* Read end is unused */
    write(fildes[1], "Hello world\n", 12);  /* Write data on pipe */
    close(fildes[1]);                       /* Child will see EOF */
    exit(EXIT_SUCCESS);
}
```

APPLICATION USAGE

None.

RATIONALE

The wording carefully avoids using the verb ‘‘to open’’ in order to avoid any implication of use of *open*(); see also *write*().

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *fcntl*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

poll — input/output multiplexing

SYNOPSIS

#include <poll.h>

int poll(struct pollfd *fds*[], nfds_t *nfds*, int *timeout*);

DESCRIPTION

The *poll*() function provides applications with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by *fds*, *poll*() shall examine the given file descriptor for the event(s) specified in *events*. The number of **pollfd** structures in the *fds* array is specified by *nfds*. The *poll*() function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.

The *fds* argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are **pollfd** structures within which *fd* specifies an open file descriptor and *events* and *revents* are bitmasks constructed by OR'ing a combination of the following event flags:

POLLIN Data other than high-priority data may be read without blocking.

For STREAMS, this flag is set in *revents* even if the message is of zero length. This flag shall be equivalent to POLLRDNORM | POLLRDBAND.

POLLRDNORM
Normal data may be read without blocking.

For STREAMS, data on priority band 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.

POLLRDBAND
Priority data may be read without blocking.

For STREAMS, data on priority bands greater than 0 may be read without blocking. This flag is set in *revents* even if the message is of zero length.

POLLPRI High-priority data may be read without blocking.

For STREAMS, this flag is set in *revents* even if the message is of zero length.

POLLOUT Normal data may be written without blocking.

For STREAMS, data on priority band 0 may be written without blocking.

POLLWRNORM
Equivalent to POLLOUT.

POLLWRBAND
Priority data may be written.

For STREAMS, data on priority bands greater than 0 may be written without blocking. If any priority band has been written to on this STREAM, this event only examines bands that have been written to at least once.

POLLERR An error has occurred on the device or stream. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.

POLLHUP A device has been disconnected, or a pipe or FIFO has been closed by the last process that had it open for writing. Once set, the hangup state of a FIFO shall persist until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed.

This event and POLLOUT are mutually-exclusive; a stream can never be writable if a hangup has occurred. However, this event and POLLIN, POLLRDNORM, POLLRDBAND, or POLLPRI are not mutually-exclusive. This flag is only valid in the *revents* bitmask; it shall be ignored in the *events* member.

POLLNVAL The specified *fd* value is invalid. This flag is only valid in the *revents* member; it shall ignored in the *events* member.

The significance and semantics of normal, priority, and high-priority data are file and device-specific.

If the value of *fd* is less than 0, *events* shall be ignored, and *revents* shall be set to 0 in that entry on return from *poll*().

In each **pollfd** structure, *poll*() shall clear the *revents* member, except that where the application requested a report on a condition by setting one of the bits of *events* listed above, *poll*() shall set the corresponding bit in *revents* if the requested condition is true. In addition, *poll*() shall set the POLLHUP, POLLERR, and POLLNVAL flag in *revents* if the condition is true, even if the application did not set the corresponding bit in *events*.

If none of the defined events have occurred on any selected file descriptor, *poll*() shall wait at least *timeout* milliseconds for an event to occur on any of the selected file descriptors. If the value of *timeout* is 0, *poll*() shall return immediately. If the value of *timeout* is −1, *poll*() shall block until a requested event occurs or until the call is interrupted.

Implementations may place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

The *poll*() function shall not be affected by the O_NONBLOCK flag.

The *poll*() function shall support regular files, terminal and pseudo-terminal devices, FIFOs, pipes, sockets and STREAMS-based files. The behavior of *poll*() on elements of *fds* that refer to other types of file is unspecified.

Regular files shall always poll TRUE for reading and writing.

A file descriptor for a socket that is listening for connections shall indicate that it is ready for reading, once connections are available. A file descriptor for a socket that is connecting asynchronously shall indicate that it is ready for writing, once a connection has been established.

Provided the application does not perform any action that results in unspecified or undefined behavior, the value of the *fd* and *events* members of each element of *fds* shall not be modified by *poll*().

RETURN VALUE

Upon successful completion, *poll*() shall return a non-negative value. A positive value indicates the total number of **pollfd** structures that have selected events (that is, those for which the *revents* member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, *poll*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *poll*() function shall fail if:

EAGAIN
The allocation of internal data structures failed but a subsequent request may succeed.

EINTR
A signal was caught during *poll*().

EINVAL
The *nfds* argument is greater than {OPEN_MAX}, or one of the *fd* members refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

The following sections are informative.

EXAMPLES

Checking for Events on a Stream

The following example opens a pair of STREAMS devices and then waits for either one to become writable. This example proceeds as follows:

1. Sets the *timeout* parameter to 500 milliseconds.
2. Opens the STREAMS devices **/dev/dev0** and **/dev/dev1**, and then polls them, specifying POLLOUT and POLLWRBAND as the events of interest.

   The STREAMS device names **/dev/dev0** and **/dev/dev1** are only examples of how STREAMS devices can be named; STREAMS naming conventions may vary among systems conforming to the POSIX.1-2008.
3. Uses the *ret* variable to determine whether an event has occurred on either of the two STREAMS. The *poll*() function is given 500 milliseconds to wait for an event to occur (if it has not occurred prior to the *poll*() call).
4. Checks the returned value of *ret*. If a positive value is returned, one of the following can be done:
   a. Priority data can be written to the open STREAM on priority bands greater than 0, because the POLLWRBAND event occurred on the open STREAM (*fds*[0] or *fds*[1]).
   b. Data can be written to the open STREAM on priority-band 0, because the POLLOUT event occurred on the open STREAM (*fds*[0] or *fds*[1]).
5. If the returned value is not a positive value, permission to write data to the open STREAM (on any priority band) is denied.
6. If the POLLHUP event occurs on the open STREAM (*fds*[0] or *fds*[1]), the device on the open STREAM has disconnected.

```
#include <stropts.h>
#include <poll.h>
...
struct pollfd fds[2];
int timeout_msecs = 500;
int ret;
  int i;

/* Open STREAMS device. */
fds[0].fd = open("/dev/dev0", ...);
fds[1].fd = open("/dev/dev1", ...);
fds[0].events = POLLOUT | POLLWRBAND;
fds[1].events = POLLOUT | POLLWRBAND;

ret = poll(fds, 2, timeout_msecs);

if (ret > 0) {
  /* An event on one of the fds has occurred. */
  for (i=0; i<2; i++) {
    if (fds[i].revents & POLLWRBAND) {
    /* Priority data may be written on device number i. */
...
    }
    if (fds[i].revents & POLLOUT) {
    /* Data may be written on device number i. */
...
    }
    if (fds[i].revents & POLLHUP) {
    /* A hangup has occurred on device number i. */
...
        }
    }
}
```

APPLICATION USAGE

None.

RATIONALE

The POLLHUP event does not occur for FIFOs just because the FIFO is not open for writing. It only occurs when the FIFO is closed by the last writer and persists until some process opens the FIFO for writing or until all read-only file descriptors for the FIFO are closed.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.6, *STREAMS*, *getmsg*( ), *pselect*( ), *putmsg*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<poll.h>**, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

popen — initiate pipe streams to or from a process

SYNOPSIS

#include <stdio.h>

FILE *popen(const char **command*, const char **mode*);

DESCRIPTION

The *popen*() function shall execute the command specified by the string *command*. It shall create a pipe between the calling program and the executed command, and shall return a pointer to a stream that can be used to either read from or write to the pipe.

The environment of the executed command shall be as if a child process were created within the *popen*() call using the *fork*() function, and the child invoked the *sh* utility using the call:

```
execl(shell path, "sh", "-c", command, (char *)0);
```

where *shell path* is an unspecified pathname for the *sh* utility.

The *popen*() function shall ensure that any streams from previous *popen*() calls that remain open in the parent process are closed in the new child process.

The *mode* argument to *popen*() is a string that specifies I/O mode:

1. If *mode* is *r*, when the child process is started, its file descriptor STDOUT_FILENO shall be the writable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*(), shall be the readable end of the pipe.
2. If *mode* is *w*, when the child process is started its file descriptor STDIN_FILENO shall be the readable end of the pipe, and the file descriptor *fileno*(*stream*) in the calling process, where *stream* is the stream pointer returned by *popen*(), shall be the writable end of the pipe.
3. If *mode* is any other value, the result is unspecified.

After *popen*(), both the parent and the child process shall be capable of executing independently before either terminates.

Pipe streams are byte-oriented.

RETURN VALUE

Upon successful completion, *popen*() shall return a pointer to an open stream that can be used to read or write to the pipe. Otherwise, it shall return a null pointer and may set *errno* to indicate the error.

ERRORS

The *popen*() function shall fail if:

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

The *popen*() function may fail if:

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

EINVAL
The *mode* argument is invalid.

The *popen*() function may also set *errno* values as described by *fork*( ) or *pipe*( ).

The following sections are informative.

EXAMPLES

Using popen( ) to Obtain a List of Files from the ls Utility

The following example demonstrates the use of *popen*() and *pclose*() to execute the command *ls** in order to obtain a list of files in the current directory:

```
#include <stdio.h>
...

FILE *fp;
int status;
char path[PATH_MAX];

fp = popen("ls *", "r");
if (fp == NULL)
   /* Handle error */;

while (fgets(path, PATH_MAX, fp) != NULL)
   printf("%s", path);

status = pclose(fp);
if (status == -1) {
   /* Error reported by pclose() */
   ...
} else {
   /* Use macros described under wait() to inspect ‘status' in order
      to determine success/failure of command executed by popen() */
   ...
}
```

APPLICATION USAGE

Since open files are shared, a mode *r* command can be used as an input filter and a mode *w* command as an output filter.

Buffered reading before opening an input filter may leave the standard input of that filter mispositioned. Similar problems with an output filter may be prevented by careful buffer flushing; for example, with *fflush*( ).

A stream opened by *popen*() should be closed by *pclose*().

The behavior of *popen*() is specified for values of *mode* of *r* and *w*. Other modes such as *rb* and *wb* might be supported by specific implementations, but these would not be portable features. Note that historical implementations of *popen*() only check to see if the first character of *mode* is *r*. Thus, a *mode* of *robert the robot* would be treated as *mode r*, and a *mode* of *anything else* would be treated as *mode w*.

If the application calls *waitpid*() or *waitid*() with a *pid* argument greater than 0, and it still has a stream that was called with *popen*() open, it must ensure that *pid* does not refer to the process started by *popen*().

To determine whether or not the environment specified in the Shell and Utilities volume of POSIX.1-2017 is present, use the function call:

```
sysconf(_SC_2_VERSION)
```

(See *sysconf*( )).

RATIONALE

The *popen*() function should not be used by programs that have set user (or group) ID privileges. The *fork*() and *exec* family of functions (except *execlp*() and *execvp*()), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

If the original and *popen*()ed processes both intend to read or write or read and write a common file, and either will be using FILE-type C functions (*fread*(), *fwrite*(), and so on), the rules for sharing file handles must be observed (see *Section 2.5.1*, *Interaction of File Descriptors and Standard I/O Streams*).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fork*( ), *pclose*( ), *pipe*( ), *sysconf*( ), *system*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

The Shell and Utilities volume of POSIX.1-2017, *sh*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_fadvise — file advisory information (**ADVANCED REALTIME**)

SYNOPSIS

#include <fcntl.h>

int posix_fadvise(int *fd*, off_t *offset*, off_t *len*, int *advice*);

DESCRIPTION

The *posix_fadvise*() function shall advise the implementation on the expected behavior of the application with respect to the data in the file associated with the open file descriptor, *fd*, starting at *offset* and continuing for *len* bytes. The specified range need not currently exist in the file. If *len* is zero, all data following *offset* is specified. The implementation may use this information to optimize handling of the specified data. The *posix_fadvise*() function shall have no effect on the semantics of other operations on the specified data, although it may affect the performance of other operations.

The advice to be applied to the data is specified by the *advice* parameter and may be one of the following values:

POSIX_FADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified data. It is the default characteristic if no advice is given for an open file.

POSIX_FADV_SEQUENTIAL
: Specifies that the application expects to access the specified data sequentially from lower offsets to higher offsets.

POSIX_FADV_RANDOM
: Specifies that the application expects to access the specified data in a random order.

POSIX_FADV_WILLNEED
: Specifies that the application expects to access the specified data in the near future.

POSIX_FADV_DONTNEED
: Specifies that the application expects that it will not access the specified data in the near future.

POSIX_FADV_NOREUSE
: Specifies that the application expects to access the specified data once and then not reuse it thereafter.

These values are defined in *<fcntl.h>*.

RETURN VALUE

Upon successful completion, *posix_fadvise*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fadvise*() function shall fail if:

EBADF
: The *fd* argument is not a valid file descriptor.

EINVAL
: The value of *advice* is invalid, or the value of *len* is less than zero.

ESPIPE
: The *fd* argument is associated with a pipe or FIFO.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *posix_fadvise*() function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_madvise( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_fallocate — file space control (**ADVANCED REALTIME**)

SYNOPSIS

#include <fcntl.h>

int posix_fallocate(int *fd*, off_t *offset*, off_t *len*);

DESCRIPTION

The *posix_fallocate*() function shall ensure that any required storage for regular file data starting at *offset* and continuing for *len* bytes is allocated on the file system storage media. If *posix_fallocate*() returns successfully, subsequent writes to the specified file data shall not fail due to the lack of free space on the file system storage media.

If the *offset+len* is beyond the current file size, then *posix_fallocate*() shall adjust the file size to *offset+len*. Otherwise, the file size shall not be changed.

It is implementation-defined whether a previous *posix_fadvise*() call influences allocation strategy.

Space allocated via *posix_fallocate*() shall be freed by a successful call to *creat*() or *open*() that truncates the size of the file. Space allocated via *posix_fallocate*() may be freed by a successful call to *ftruncate*() that reduces the file size to a size smaller than *offset+len*.

RETURN VALUE

Upon successful completion, *posix_fallocate*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_fallocate*() function shall fail if:

EBADF
The *fd* argument is not a valid file descriptor.

EBADF
The *fd* argument references a file that was opened without write permission.

EFBIG
The value of *offset+len* is greater than the maximum file size.

EINTR
A signal was caught during execution.

EINVAL
The *len* argument is less than zero, or the *offset* argument is less than zero, or the underlying file system does not support this operation.

EIO An I/O error occurred while reading from or writing to a file system.

ENODEV
The *fd* argument does not refer to a regular file.

ENOSPC
There is insufficient free space remaining on the file system storage media.

ESPIPE
The *fd* argument is associated with a pipe or FIFO.

The *posix_fallocate*() function may fail if:

EINVAL
The *len* argument is zero.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
The *posix_fallocate*() function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
creat( ), *ftruncate*( ), *open*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_madvise — memory advisory information and alignment control (**ADVANCED REALTIME**)

SYNOPSIS

#include <sys/mman.h>

int posix_madvise(void **addr*, size_t *len*, int *advice*);

DESCRIPTION

The *posix_madvise*() function shall advise the implementation on the expected behavior of the application with respect to the data in the memory starting at address *addr*, and continuing for *len* bytes. The implementation may use this information to optimize handling of the specified data. The *posix_madvise*() function shall have no effect on the semantics of access to memory in the specified range, although it may affect the performance of access.

The implementation may require that *addr* be a multiple of the page size, which is the value returned by *sysconf*() when the name value _SC_PAGESIZE is used.

The advice to be applied to the memory range is specified by the *advice* parameter and may be one of the following values:

POSIX_MADV_NORMAL
: Specifies that the application has no advice to give on its behavior with respect to the specified range. It is the default characteristic if no advice is given for a range of memory.

POSIX_MADV_SEQUENTIAL
: Specifies that the application expects to access the specified range sequentially from lower addresses to higher addresses.

POSIX_MADV_RANDOM
: Specifies that the application expects to access the specified range in a random order.

POSIX_MADV_WILLNEED
: Specifies that the application expects to access the specified range in the near future.

POSIX_MADV_DONTNEED
: Specifies that the application expects that it will not access the specified range in the near future.

These values are defined in the *<sys/mman.h>* header.

RETURN VALUE

Upon successful completion, *posix_madvise*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_madvise*() function shall fail if:

EINVAL
: The value of *advice* is invalid.

ENOMEM
: Addresses in the range starting at *addr* and continuing for *len* bytes are partly or completely outside the range allowed for the address space of the calling process.

The *posix_madvise*() function may fail if:

EINVAL
: The value of *addr* is not a multiple of the value returned by *sysconf*() when the name value _SC_PAGESIZE is used.

EINVAL
The value of *len* is zero.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *posix_madvise*() function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *posix_fadvise*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_mem_offset — find offset and length of a mapped typed memory block (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int posix_mem_offset(const void *restrict addr, size_t len,
    off_t *restrict off, size_t *restrict contig_len,
    int *restrict fildes);
```

DESCRIPTION

The *posix_mem_offset*() function shall return in the variable pointed to by *off* a value that identifies the offset (or location), within a memory object, of the memory block currently mapped at *addr*. The function shall return in the variable pointed to by *fildes*, the descriptor used (via *mmap*()) to establish the mapping which contains *addr*. If that descriptor was closed since the mapping was established, the returned value of *fildes* shall be −1. The *len* argument specifies the length of the block of the memory object the user wishes the offset for; upon return, the value pointed to by *contig_len* shall equal either *len*, or the length of the largest contiguous block of the memory object that is currently mapped to the calling process starting at *addr*, whichever is smaller.

If the memory object mapped at *addr* is a typed memory object, then if the *off* and *contig_len* values obtained by calling *posix_mem_offset*() are used in a call to *mmap*() with a file descriptor that refers to the same memory pool as *fildes* (either through the same port or through a different port), and that was opened with neither the POSIX_TYPED_MEM_ALLOCATE nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag, the typed memory area that is mapped shall be exactly the same area that was mapped at *addr* in the address space of the process that called *posix_mem_offset*().

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is implementation-defined.

RETURN VALUE

Upon successful completion, the *posix_mem_offset*() function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_mem_offset*() function shall fail if:

EACCES
The process has not mapped a memory object supported by this function at the given address *addr*.

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mmap( ), *posix_typed_mem_open*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_memalign — aligned memory allocation (**ADVANCED REALTIME**)

SYNOPSIS

#include <stdlib.h>

int posix_memalign(void ***memptr*, size_t *alignment*, size_t *size*);

DESCRIPTION

The *posix_memalign*() function shall allocate *size* bytes aligned on a boundary specified by *alignment*, and shall return a pointer to the allocated memory in *memptr*. The value of *alignment* shall be a power of two multiple of *sizeof* (**void ***).

Upon successful completion, the value pointed to by *memptr* shall be a multiple of *alignment*.

If the size of the space requested is 0, the behavior is implementation-defined: either a null pointer shall be returned in *memptr*, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the the value returned in *memptr* is used to access an object.

The *free*() function shall deallocate memory that has previously been allocated by *posix_memalign*().

RETURN VALUE

Upon successful completion, *posix_memalign*() shall return zero; otherwise, an error number shall be returned to indicate the error and the contents of *memptr* shall either be left unmodified or be set to a null pointer.

If *size* is 0, either:

- *posix_memalign*() shall not attempt to allocate any space, in which case either an implementation-defined error number shall be returned, or zero shall be returned with a null pointer returned in *memptr*, or
- *posix_memalign*() shall attempt to allocate some space and, if the allocation succeeds, zero shall be returned and a pointer to the allocated space shall be returned in *memptr*. The application shall ensure that the pointer is not used to access an object.

ERRORS

The *posix_memalign*() function shall fail if:

EINVAL
The value of the alignment parameter is not a power of two multiple of *sizeof* (**void ***).

ENOMEM
There is insufficient memory available with the requested alignment.

The following sections are informative.

EXAMPLES

The following example shows how applications can obtain consistent behavior on error by setting **memptr* to be a null pointer before calling *posix_memalign*().

```
void *ptr = NULL;
...
//do some work, which might goto error
if (posix_memalign(&ptr, align, size))
    goto error;

//do some more work, which might goto error
...
```

```
error:
   free(ptr);
   //more cleanup;
```

APPLICATION USAGE

The *posix_memalign*() function is part of the Advisory Information option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *malloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_openpt — open a pseudo-terminal device

SYNOPSIS

```
#include <stdlib.h>
#include <fcntl.h>

int posix_openpt(int oflag);
```

DESCRIPTION

The *posix_openpt*() function shall establish a connection between a master device for a pseudo-terminal and a file descriptor. The file descriptor shall be allocated as described in *Section 2.14*, *File Descriptor Allocation* and can be used by other I/O functions that refer to that pseudo-terminal.

The file status flags and file access modes of the open file description shall be set according to the value of *oflag*.

Values for *oflag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

O_RDWR	Open for reading and writing.
O_NOCTTY	If set *posix_openpt*() shall not cause the terminal device to become the controlling terminal for the process.

The behavior of other values for the *oflag* argument is unspecified.

RETURN VALUE

Upon successful completion, the *posix_openpt*() function shall open a file descriptor for a master pseudo-terminal device and return a non-negative integer representing the file descriptor. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *posix_openpt*() function shall fail if:

EMFILE
All file descriptors available to the process are currently open.

ENFILE
The maximum allowable number of files is currently open in the system.

The *posix_openpt*() function may fail if:

EINVAL
The value of *oflag* is not valid.

EAGAIN
Out of pseudo-terminal resources.

ENOSR
Out of STREAMS resources.

The following sections are informative.

EXAMPLES

Opening a Pseudo-Terminal and Returning the Name of the Slave Device and a File Descriptor

```
#include <fcntl.h>
#include <stdio.h>

int masterfd, slavefd;
char *slavedevice;
```

```
masterfd = posix_openpt(O_RDWR|O_NOCTTY);

if (masterfd == -1
    || grantpt (masterfd) == -1
    || unlockpt (masterfd) == -1
    || (slavedevice = ptsname (masterfd)) == NULL)
    return -1;

printf("slave device is: %s\n", slavedevice);

slavefd = open(slavedevice, O_RDWR|O_NOCTTY);
if (slavefd < 0)
    return -1;
```

APPLICATION USAGE

This function is a method for portably obtaining a file descriptor of a master terminal device for a pseudo-terminal. The *grantpt*() and *ptsname*() functions can be used to manipulate mode and ownership permissions, and to obtain the name of the slave device, respectively.

RATIONALE

The standard developers considered the matter of adding a special device for cloning master pseudo-terminals: the **/dev/ptmx** device. However, consensus could not be reached, and it was felt that adding a new function would permit other implementations. The *posix_openpt*() function is designed to complement the *grantpt*(), *ptsname*(), and *unlockpt*() functions.

On implementations supporting the **/dev/ptmx** clone device, opening the master device of a pseudo-terminal is simply:

```
mfdp = open("/dev/ptmx", oflag );
if (mfdp < 0)
    return -1;
```

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *grantpt*( ), *open*( ), *ptsname*( ), *unlockpt*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn, posix_spawnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn(pid_t *restrict pid, const char *restrict path,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

DESCRIPTION

The *posix_spawn*() and *posix_spawnp*() functions shall create a new process (child process) from the specified process image. The new process image shall be constructed from a regular executable file called the new process image file.

When a C program is executed as the result of this call, it shall be entered as a C-language function call as follows:

```
    int main(int argc, char *argv[]);
```

where *argc* is the argument count and *argv* is an array of character pointers to the arguments themselves. In addition, the following variable:

```
    extern char **environ;
```

shall be initialized as a pointer to an array of character pointers to the environment strings.

The argument *argv* is an array of character pointers to null-terminated strings. The last member of this array shall be a null pointer and is not counted in *argc*. These strings constitute the argument list available to the new process image. The value in *argv*[0] should point to a filename string that is associated with the process image being started by the *posix_spawn*() or *posix_spawnp*() function.

The argument *envp* is an array of character pointers to null-terminated strings. These strings constitute the environment for the new process image. The environment array is terminated by a null pointer.

The number of bytes available for the combined argument and environment lists of the child process is {ARG_MAX}. The implementation shall specify in the system documentation (see the Base Definitions volume of POSIX.1-2017, *Chapter 2*, *Conformance*) whether any list overhead, such as length words, null terminators, pointers, or alignment bytes, is included in this total.

The *path* argument to *posix_spawn*() is a pathname that identifies the new process image file to execute.

The *file* parameter to *posix_spawnp*() shall be used to construct a pathname that identifies the new process image file. If the *file* parameter contains a <slash> character, the *file* parameter shall be used as the pathname for the new process image file. Otherwise, the path prefix for this file shall be obtained by a search of the directories passed as the environment variable *PATH* (see the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*). If this environment variable is not defined, the results of the search are implementation-defined.

If *file_actions* is a null pointer, then file descriptors open in the calling process shall remain open in the child process, except for those whose close-on-*exec* flag FD_CLOEXEC is set (see *fcntl*( )). For those file descriptors that remain open, the child process shall not inherit any file locks, but all remaining attributes of the corresponding open file descriptions (see *fcntl*( )), shall remain unchanged.

If *file_actions* is not NULL, then the file descriptors open in the child process shall be those open in the calling process as modified by the spawn file actions object pointed to by *file_actions* and the FD_CLOEXEC flag of each remaining open file descriptor after the spawn file actions have been processed. The effective order of processing the spawn file actions shall be:

1. The set of open file descriptors for the child process shall initially be the same set as is open for the calling process. The child process shall not inherit any file locks, but all remaining attributes of the corresponding open file descriptions (see *fcntl*( )), shall remain unchanged.
2. The signal mask, signal default actions, and the effective user and group IDs for the child process shall be changed as specified in the attributes object referenced by *attrp*.
3. The file actions specified by the spawn file actions object shall be performed in the order in which they were added to the spawn file actions object.
4. Any file descriptor that has its FD_CLOEXEC flag set (see *fcntl*( )) shall be closed.

If file descriptor 0, 1, or 2 would otherwise be closed in the new process image created by *posix_spawn*() or *posix_spawnp*(), implementations may open an unspecified file for the file descriptor in the new process image. If a standard utility or a conforming application is executed with file descriptor 0 not open for reading or with file descriptor 1 or 2 not open for writing, the environment in which the utility or application is executed shall be deemed non-conforming, and consequently the utility or application might not behave as described in this standard.

The **posix_spawnattr_t** spawn attributes object type is defined in <*spawn.h*>. It shall contain at least the attributes defined below.

If the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is non-zero, then the child's process group shall be as specified in the *spawn-pgroup* attribute of the object referenced by *attrp*.

As a special case, if the POSIX_SPAWN_SETPGROUP flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, and the *spawn-pgroup* attribute of the same object is set to zero, then the child shall be in a new process group with a process group ID equal to its process ID.

If the POSIX_SPAWN_SETPGROUP flag is not set in the *spawn-flags* attribute of the object referenced by *attrp*, the new child process shall inherit the parent's process group.

If the POSIX_SPAWN_SETSCHEDPARAM flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, but POSIX_SPAWN_SETSCHEDULER is not set, the new process image shall initially have the scheduling policy of the calling process with the scheduling parameters specified in the *spawn-schedparam* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSCHEDULER flag is set in the *spawn-flags* attribute of the object referenced by *attrp* (regardless of the setting of the POSIX_SPAWN_SETSCHEDPARAM flag), the new process image shall initially have the scheduling policy specified in the *spawn-schedpolicy* attribute of the object referenced by *attrp* and the scheduling parameters specified in the *spawn-schedparam* attribute of the same object.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* governs the effective user ID of the child process. If this flag is not set, the child process shall inherit the effective user ID of the parent process. If this flag is set, the effective user ID of the child process shall be reset to the parent's real user ID. In either case, if the set-user-ID mode bit of the new process image file is set, the effective user ID of the child process shall become that file's owner ID before the new process image begins execution.

The POSIX_SPAWN_RESETIDS flag in the *spawn-flags* attribute of the object referenced by *attrp* also governs the effective group ID of the child process. If this flag is not set, the child process shall inherit the effective group ID of the parent process. If this flag is set, the effective group ID of the child process shall be reset to the parent's real group ID. In either case, if the set-group-ID mode bit of the new process image file is set, the effective group ID of the child process shall become that file's group ID before the new process image begins execution.

If the POSIX_SPAWN_SETSIGMASK flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the child process shall initially have the signal mask specified in the *spawn-sigmask* attribute of the object referenced by *attrp*.

If the POSIX_SPAWN_SETSIGDEF flag is set in the *spawn-flags* attribute of the object referenced by *attrp*, the signals specified in the *spawn-sigdefault* attribute of the same object shall be set to their default actions in the child process. Signals set to the default action in the parent process shall be set to the default action in the child process.

Signals set to be caught by the calling process shall be set to the default action in the child process.

Except for SIGCHLD, signals set to be ignored by the calling process image shall be set to be ignored by the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn-flags* attribute of the object referenced by *attrp* and the signals being indicated in the *spawn-sigdefault* attribute of the object referenced by *attrp*.

If the SIGCHLD signal is set to be ignored by the calling process, it is unspecified whether the SIGCHLD signal is set to be ignored or to the default action in the child process, unless otherwise specified by the POSIX_SPAWN_SETSIGDEF flag being set in the *spawn_flags* attribute of the object referenced by *attrp* and the SIGCHLD signal being indicated in the *spawn_sigdefault* attribute of the object referenced by *attrp*.

If the value of the *attrp* pointer is NULL, then the default values are used.

All process attributes, other than those influenced by the attributes set in the object referenced by *attrp* as specified above or by the file descriptor manipulations specified in *file_actions*, shall appear in the new process image as though *fork*() had been called to create a child process and then a member of the *exec* family of functions had been called by the child process to execute the new process image.

It is implementation-defined whether the fork handlers are run when *posix_spawn*() or *posix_spawnp*() is called.

RETURN VALUE

Upon successful completion, *posix_spawn*() and *posix_spawnp*() shall return the process ID of the child process to the parent process, in the variable pointed to by a non-NULL *pid* argument, and shall return zero as the function return value. Otherwise, no child process shall be created, the value stored into the variable pointed to by a non-NULL *pid* is unspecified, and an error number shall be returned as the function return value to indicate the error. If the *pid* argument is a null pointer, the process ID of the child is not returned to the caller.

ERRORS

These functions may fail if:

EINVAL
The value specified by *file_actions* or *attrp* is invalid.

If this error occurs after the calling process successfully returns from the *posix_spawn*() or *posix_spawnp*() function, the child process may exit with exit status 127.

If *posix_spawn*() or *posix_spawnp*() fail for any of the reasons that would cause *fork*() or one of the *exec* family of functions to fail, an error value shall be returned as described by *fork*() and *exec*, respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute of the object referenced by *attrp*, and *posix_spawn*() or *posix_spawnp*() fails while changing the child's process group, an error value shall be returned as described by *setpgid*() (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If POSIX_SPAWN_SETSCHEDPARAM is set and POSIX_SPAWN_SETSCHEDULER is not set in the *spawn-flags* attribute of the object referenced by *attrp*, then if *posix_spawn*() or *posix_spawnp*() fails for any of the reasons that would cause *sched_setparam*() to fail, an error value shall be returned as described by *sched_setparam*() (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute of the object referenced by *attrp*, and if *posix_spawn*() or *posix_spawnp*() fails for any of the reasons that would cause *sched_setscheduler*() to fail, an error value shall be returned as described by *sched_setscheduler*() (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127).

If the *file_actions* argument is not NULL, and specifies any *close*, *dup2*, or *open* actions to be performed, and if *posix_spawn*() or *posix_spawnp*() fails for any of the reasons that would cause *close*(), *dup2*(), or *open*() to fail, an error value shall be returned as described by *close*(), *dup2*(), and *open*(), respectively (or, if the error occurs after the calling process successfully returns, the child process shall exit with exit status 127). An open file action may, by itself, result in any of the errors described by *close*() or *dup2*(), in addition to those described by *open*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

See also the APPLICATION USAGE section for *exec*.

RATIONALE

The *posix_spawn*() function and its close relation *posix_spawnp*() have been introduced to overcome the following perceived difficulties with *fork*(): the *fork*() function is difficult or impossible to implement without swapping or dynamic address translation.

* Swapping is generally too slow for a realtime environment.

* Dynamic address translation is not available everywhere that POSIX might be useful.

* Processes are too useful to simply option out of POSIX whenever it must run without address translation or other MMU services.

Thus, POSIX needs process creation and file execution primitives that can be efficiently implemented without address translation or other MMU services.

The *posix_spawn*() function is implementable as a library routine, but both *posix_spawn*() and *posix_spawnp*() are designed as kernel operations. Also, although they may be an efficient replacement for many *fork*()/*exec* pairs, their goal is to provide useful process creation primitives for systems that have difficulty with *fork*(), not to provide drop-in replacements for *fork*()/*exec*.

This view of the role of *posix_spawn*() and *posix_spawnp*() influenced the design of their API. It does not attempt to provide the full functionality of *fork*()/*exec* in which arbitrary user-specified operations of any sort are permitted between the creation of the child process and the execution of the new process image; any attempt to reach that level would need to provide a programming language as parameters. Instead, *posix_spawn*() and *posix_spawnp*() are process creation primitives like the *Start_Process* and *Start_Process_Search* Ada language bindings package *POSIX_Process_Primitives* and also like those in many operating systems that are not UNIX systems, but with some POSIX-specific additions.

To achieve its coverage goals, *posix_spawn*() and *posix_spawnp*() have control of six types of inheritance: file descriptors, process group ID, user and group ID, signal mask, scheduling, and whether each signal ignored in the parent will remain ignored in the child, or be reset to its default action in the child.

Control of file descriptors is required to allow an independently written child process image to access data streams opened by and even generated or read by the parent process without being specifically coded to know which parent files and file descriptors are to be used. Control of the process group ID is required to control how the job control of the child process relates to that of the parent.

Control of the signal mask and signal defaulting is sufficient to support the implementation of *system*(). Although support for *system*() is not explicitly one of the goals for *posix_spawn*() and *posix_spawnp*(), it is covered under the "at least 50%" coverage goal.

The intention is that the normal file descriptor inheritance across *fork*(), the subsequent effect of the specified spawn file actions, and the normal file descriptor inheritance across one of the *exec* family of functions should fully specify open file inheritance. The implementation need make no decisions regarding the set of open file descriptors when the child process image begins execution, those decisions having already been made by the caller and expressed as the set of open file descriptors and their FD_CLOEXEC flags at the time of the call and the spawn file actions object specified in the call. We have been assured that in cases where the POSIX *Start_Process* Ada primitives have been implemented in a library, this method of controlling file descriptor inheritance may be implemented very easily.

We can identify several problems with *posix_spawn*() and *posix_spawnp*(), but there does not appear to be a solution that introduces fewer problems. Environment modification for child process attributes not specifiable via the *attrp* or *file_actions* arguments must be done in the parent process, and since the parent generally wants to save its context, it is more costly than similar functionality with *fork*()/*exec*. It is also complicated to modify the environment of a multi-threaded process temporarily, since all threads must agree when it is safe for the environment to be changed. However, this cost is only borne by those invocations of *posix_spawn*() and *posix_spawnp*() that use the additional functionality. Since extensive modifications are not the usual case, and are particularly unlikely in time-critical code, keeping much of the environment control out of *posix_spawn*() and *posix_spawnp*() is appropriate design.

The *posix_spawn*() and *posix_spawnp*() functions do not have all the power of *fork*()/*exec*. This is to be expected. The *fork*() function is a wonderfully powerful operation. We do not expect to duplicate its functionality in a simple, fast function with no special hardware requirements. It is worth noting that *posix_spawn*() and *posix_spawnp*() are very similar to the process creation operations on many operating systems that are not UNIX systems.

Requirements

The requirements for *posix_spawn*() and *posix_spawnp*() are:

* They must be implementable without an MMU or unusual hardware.
* They must be compatible with existing POSIX standards.

Additional goals are:

* They should be efficiently implementable.
* They should be able to replace at least 50% of typical executions of *fork*().
* A system with *posix_spawn*() and *posix_spawnp*() and without *fork*() should be useful, at least for realtime applications.
* A system with *fork*() and the *exec* family should be able to implement *posix_spawn*() and *posix_spawnp*() as library routines.

Two-Syntax

POSIX *exec* has several calling sequences with approximately the same functionality. These appear to be required for compatibility with existing practice. Since the existing practice for the *posix_spawn**( ) functions is otherwise substantially unlike POSIX, we feel that simplicity outweighs compatibility. There are, therefore, only two names for the *posix_spawn**( ) functions.

The parameter list does not differ between *posix_spawn*() and *posix_spawnp*(); *posix_spawnp*() interprets the second parameter more elaborately than *posix_spawn*().

Compatibility with POSIX.5 (Ada)

The *Start_Process* and *Start_Process_Search* procedures from the *POSIX_Process_Primitives* package from the Ada language binding to POSIX.1 encapsulate *fork*() and *exec* functionality in a manner similar to that of *posix_spawn*() and *posix_spawnp*(). Originally, in keeping with our simplicity goal, the standard developers had limited the capabilities of *posix_spawn*() and *posix_spawnp*() to a subset of the capabilities of *Start_Process* and *Start_Process_Search*; certain non-default capabilities were not supported. However, based on suggestions by the ballot group to improve file descriptor mapping or drop it, and on the advice of an Ada Language Bindings working group member, the standard developers decided that *posix_spawn*() and *posix_spawnp*() should be sufficiently powerful to implement *Start_Process* and *Start_Process_Search*. The rationale is that if the Ada language binding to such a primitive had already been approved as an IEEE standard, there can be little justification for not approving the functionally-equivalent parts of a C binding. The only three capabilities provided by *posix_spawn*() and *posix_spawnp*() that are not provided by *Start_Process* and *Start_Process_Search* are optionally specifying the child's process group ID, the set of signals to be reset to default signal handling in the child process, and the child's scheduling policy and parameters.

For the Ada language binding for *Start_Process* to be implemented with *posix_spawn*(), that binding would need to explicitly pass an empty signal mask and the parent's environment to *posix_spawn*() whenever the caller of *Start_Process* allowed these arguments to default, since *posix_spawn*() does not provide such defaults. The ability of *Start_Process* to mask user-specified signals during its execution is functionally unique to the Ada language binding and must be dealt with in the binding separately from the call to *posix_spawn*().

Process Group

The process group inheritance field can be used to join the child process with an existing process group. By assigning a value of zero to the *spawn-pgroup* attribute of the object referenced by *attrp*, the *setpgid*() mechanism will place the child process in a new process group.

Threads

Without the *posix_spawn*() and *posix_spawnp*() functions, systems without address translation can still use threads to give an abstraction of concurrency. In many cases, thread creation suffices, but it is not always a good substitute. The *posix_spawn*() and *posix_spawnp*() functions are considerably ``heavier'' than thread creation. Processes have several important attributes that threads do not. Even without address translation, a process may have base-and-bound memory protection. Each process has a process environment including security attributes and file capabilities, and powerful scheduling attributes. Processes abstract the behavior of non-uniform-memory-architecture multi-processors better than threads, and they are more convenient to use for activities that are not closely linked.

The *posix_spawn*() and *posix_spawnp*() functions may not bring support for multiple processes to every configuration. Process creation is not the only piece of operating system support required to support multiple processes. The total cost of support for multiple processes may be quite high in some circumstances. Existing practice shows that support for multiple processes is uncommon and threads are common among ``tiny kernels''. There should, therefore, probably continue to be AEPs for operating systems with only one process.

Asynchronous Error Notification

A library implementation of *posix_spawn*() or *posix_spawnp*() may not be able to detect all possible errors before it forks the child process. POSIX.1-2008 provides for an error indication returned from a child process which could not successfully complete the spawn operation via a special exit status which may be detected using the status value returned by *wait*(), *waitid*(), and *waitpid*().

The *stat_val* interface and the macros used to interpret it are not well suited to the purpose of returning API errors, but they are the only path available to a library implementation. Thus, an implementation may cause the child process to exit with exit status 127 for any error detected during the spawn process after the *posix_spawn*() or *posix_spawnp*() function has successfully returned.

The standard developers had proposed using two additional macros to interpret *stat_val*. The first, WIFSPAWNFAIL, would have detected a status that indicated that the child exited because of an error detected during the *posix_spawn*() or *posix_spawnp*() operations rather than during actual execution of the child process image; the second, WSPAWNERRNO, would have extracted the error value if WIFSPAWNFAIL indicated a failure. Unfortunately, the ballot group strongly opposed this because it would make a library implementation of *posix_spawn*() or *posix_spawnp*() dependent on kernel modifications to *waitpid*() to be able to embed special information in *stat_val* to indicate a spawn failure.

The 8 bits of child process exit status that are guaranteed by POSIX.1-2008 to be accessible to the waiting parent process are insufficient to disambiguate a spawn error from any other kind of error that may be returned by an arbitrary process image. No other bits of the exit status are required to be visible in *stat_val*, so these macros could not be strictly implemented at the library level. Reserving an exit status of 127 for such spawn errors is consistent with the use of this value by *system*() and *popen*() to signal failures in these operations that occur after the function has returned but before a shell is able to execute. The exit status of 127 does not uniquely identify this class of error, nor does it provide any detailed information on the nature of the failure. Note that a kernel implementation of *posix_spawn*() or *posix_spawnp*() is permitted (and encouraged) to return any possible error as the function value, thus providing more detailed failure information to the parent process.

Thus, no special macros are available to isolate asynchronous *posix_spawn*() or *posix_spawnp*() errors. Instead, errors detected by the *posix_spawn*() or *posix_spawnp*() operations in the context of the child process before the new process image executes are reported by setting the child's exit status to 127. The calling process may use the WIFEXITED and WEXITSTATUS macros on the *stat_val* stored by the *wait*() or *waitpid*() functions to detect spawn failures to the extent that other status values with which the child process image may exit (before the parent can conclusively determine that the child process image has begun execution) are distinct from exit status 127.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *chmod*( ), *close*( ), *dup*( ), *exec*, *exit*( ), *fcntl*( ), *fork*( ), *fstatat*( ), *kill*( ), *open*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_destroy*( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( ), *sched_setparam*( ), *sched_setscheduler*( ), *setpgid*( ), *setuid*( ), *times*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn_file_actions_addclose, posix_spawn_file_actions_addopen — add close or open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_addclose(posix_spawn_file_actions_t
    *file_actions, int fildes);
int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
    *restrict file_actions, int fildes,
    const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

These functions shall add or delete a close or open action to a spawn file actions object.

A spawn file actions object is of type **posix_spawn_file_actions_t** (defined in <*spawn.h*>) and is used to specify a series of actions to be performed by a *posix_spawn*() or *posix_spawnp*() operation in order to arrive at the set of open file descriptors for the child process given the set of open file descriptors of the parent. POSIX.1-2008 does not define comparison or assignment operators for the type **posix_spawn_file_actions_t**.

A spawn file actions object, when passed to *posix_spawn*() or *posix_spawnp*(), shall specify how the set of open file descriptors in the calling process is transformed into a set of potentially open file descriptors for the spawned process. This transformation shall be as if the specified sequence of actions was performed exactly once, in the context of the spawned process (prior to execution of the new process image), in the order in which the actions were added to the object; additionally, when the new process image is executed, any file descriptor (from this new set) which has its FD_CLOEXEC flag set shall be closed (see *posix_spawn*( )).

The *posix_spawn_file_actions_addclose*() function shall add a *close* action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be closed (as if *close*(*fildes*) had been called) when a new process is spawned using this file actions object.

The *posix_spawn_file_actions_addopen*() function shall add an *open* action to the object referenced by *file_actions* that shall cause the file named by *path* to be opened (as if *open*(*path*, *oflag*, *mode*) had been called, and the returned file descriptor, if not *fildes*, had been changed to *fildes*) when a new process is spawned using this file actions object. If *fildes* was already an open file descriptor, it shall be closed before the new file is opened.

The string described by *path* shall be copied by the *posix_spawn_file_actions_addopen*() function.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_addopen*() function shall fail if:

EBADF
The value specified by *fildes* is negative or greater than or equal to {OPEN_MAX}.

The *posix_spawn_file_actions_addclose*() function shall fail if:

EBADF
The value specified by *fildes* is negative.

These functions may fail if:

EINVAL
The value specified by *file_actions* is invalid.

ENOMEM
Insufficient memory exists to add to the spawn file actions object.

It shall not be considered an error for the *fildes* argument passed to these functions to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error will be detected when the associated file actions object is later used during a *posix_spawn*() or *posix_spawnp*() operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *posix_spawn_file_actions_addclose*() with an arbitrary integer risks non-conforming behavior, and this function can only portably be used to close file descriptor values that the application has obtained through explicit actions, or for the three file descriptors corresponding to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process, an application should consider opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

RATIONALE

A spawn file actions object may be initialized to contain an ordered sequence of *close*(), *dup2*(), and *open*() operations to be used by *posix_spawn*() or *posix_spawnp*() to arrive at the set of open file descriptors inherited by the spawned process from the set of open file descriptors in the parent at the time of the *posix_spawn*() or *posix_spawnp*() call. It had been suggested that the *close*() and *dup2*() operations alone are sufficient to rearrange file descriptors, and that files which need to be opened for use by the spawned process can be handled either by having the calling process open them before the *posix_spawn*() or *posix_spawnp*() call (and close them after), or by passing pathnames to the spawned process (in *argv*) so that it may open them itself. The standard developers recommend that applications use one of these two methods when practical, since detailed error status on a failed open operation is always available to the application this way. However, the standard developers feel that allowing a spawn file actions object to specify open operations is still appropriate because:

1. It is consistent with equivalent POSIX.5 (Ada) functionality.
2. It supports the I/O redirection paradigm commonly employed by POSIX programs designed to be invoked from a shell. When such a program is the child process, it may not be designed to open files on its own.
3. It allows file opens that might otherwise fail or violate file ownership/access rights if executed by the parent process.

Regarding 2. above, note that the spawn open file action provides to *posix_spawn*() and *posix_spawnp*() the same capability that the shell redirection operators provide to *system*(), only without the intervening execution of a shell; for example:

```
system ("myprog <file1 3<file2");
```

Regarding 3. above, note that if the calling process needs to open one or more files for access by the spawned process, but has insufficient spare file descriptors, then the open action is necessary to allow the *open*() to occur in the context of the child process after other file descriptors have been closed (that must remain open in the parent).

Additionally, if a parent is executed from a file having a "set-user-id" mode bit set and the POSIX_SPAWN_RESETIDS flag is set in the spawn attributes, a file created within the parent process will (possibly incorrectly) have the parent's effective user ID as its owner, whereas a file created via an *open*() action during *posix_spawn*() or *posix_spawnp*() will have the parent's real ID as its owner; and an open by the parent process may successfully open a file to which the real user should not have access or fail to open a file to which the real user should have access.

File Descriptor Mapping

The standard developers had originally proposed using an array which specified the mapping of child file descriptors back to those of the parent. It was pointed out by the ballot group that it is not possible to reshuffle file descriptors arbitrarily in a library implementation of *posix_spawn*() or *posix_spawnp*() without provision for one or more spare file descriptor entries (which simply may not be available). Such an array requires that an implementation develop a complex strategy to achieve the desired mapping without inadvertently closing the wrong file descriptor at the wrong time.

It was noted by a member of the Ada Language Bindings working group that the approved Ada Language *Start_Process* family of POSIX process primitives use a caller-specified set of file actions to alter the normal *fork*()/*exec* semantics for inheritance of file descriptors in a very flexible way, yet no such problems exist because the burden of determining how to achieve the final file descriptor mapping is completely on the application. Furthermore, although the file actions interface appears frightening at first glance, it is actually quite simple to implement in either a library or the kernel.

The *posix_spawn_file_actions_addclose*() function is not required to check whether the file descriptor is less than {OPEN_MAX} because on some implementations {OPEN_MAX} reflects the RLIMIT_NOFILE soft limit and therefore calling *setrlimit*() to reduce this limit can result in an {OPEN_MAX} value less than or equal to an already open file descriptor. Applications need to be able to close such file descriptors on spawn. On implementations where {OPEN_MAX} does not change, it is recommended that *posix_spawn_file_actions_addclose*() should return **[EBADF]** if *fildes* is greater than or equal to {OPEN_MAX}.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *dup*( ), *open*( ), *posix_spawn*( ), *posix_spawn_file_actions_adddup2*( ), *posix_spawn_file_actions_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn_file_actions_adddup2 — add dup2 action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_adddup2(posix_spawn_file_actions_t
   *file_actions, int fildes, int newfildes);
```

DESCRIPTION

The *posix_spawn_file_actions_adddup2*() function shall add a *dup2*() action to the object referenced by *file_actions* that shall cause the file descriptor *fildes* to be duplicated as *newfildes* (as if *dup2*(*fildes*, *newfildes*) had been called) when a new process is spawned using this file actions object.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

RETURN VALUE

Upon successful completion, the *posix_spawn_file_actions_adddup2*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_adddup2*() function shall fail if:

EBADF
The value specified by *fildes* or *newfildes* is negative or greater than or equal to {OPEN_MAX}.

ENOMEM
Insufficient memory exists to add to the spawn file actions object.

The *posix_spawn_file_actions_adddup2*() function may fail if:

EINVAL
The value specified by *file_actions* is invalid.

It shall not be considered an error for the *fildes* argument passed to the *posix_spawn_file_actions_adddup2*() function to specify a file descriptor for which the specified operation could not be performed at the time of the call. Any such error will be detected when the associated file actions object is later used during a *posix_spawn*() or *posix_spawnp*() operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *posix_spawn_file_actions_adddup2*() function is part of the Spawn option and need not be provided on all implementations.

Implementations may use file descriptors that must be inherited into child processes for the child process to remain conforming, such as for message catalog or tracing purposes. Therefore, an application that calls *posix_spawn_file_actions_adddup2*() with an arbitrary integer for *newfildes* risks non-conforming behavior, and this function can only portably be used to overwrite file descriptor values that the application has obtained through explicit actions, or for the three file descriptors corresponding to the standard file streams. In order to avoid a race condition of leaking an unintended file descriptor into a child process, an application should consider opening all file descriptors with the FD_CLOEXEC bit set unless the file descriptor is intended to be inherited across *exec*.

RATIONALE

Refer to the RATIONALE section in *posix_spawn_file_actions_addclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

dup( ), *posix_spawn*( ), *posix_spawn_file_actions_addclose*( ), *posix_spawn_file_actions_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn_file_actions_addopen — add open action to spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_addopen(posix_spawn_file_actions_t
   *restrict file_actions, int fildes,
   const char *restrict path, int oflag, mode_t mode);
```

DESCRIPTION

Refer to *posix_spawn_file_actions_addclose*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawn_file_actions_destroy, posix_spawn_file_actions_init — destroy and initialize spawn file actions object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawn_file_actions_destroy(posix_spawn_file_actions_t
   *file_actions);
int posix_spawn_file_actions_init(posix_spawn_file_actions_t
   *file_actions);
```

DESCRIPTION

The *posix_spawn_file_actions_destroy*() function shall destroy the object referenced by *file_actions*; the object becomes, in effect, uninitialized. An implementation may cause *posix_spawn_file_actions_destroy*() to set the object referenced by *file_actions* to an invalid value. A destroyed spawn file actions object can be reinitialized using *posix_spawn_file_actions_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

The *posix_spawn_file_actions_init*() function shall initialize the object referenced by *file_actions* to contain no file actions for *posix_spawn*() or *posix_spawnp*() to perform.

A spawn file actions object is as defined in *posix_spawn_file_actions_addclose*( ).

The effect of initializing an already initialized spawn file actions object is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawn_file_actions_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the spawn file actions object.

The *posix_spawn_file_actions_destroy*() function may fail if:

EINVAL
The value specified by *file_actions* is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

Refer to the RATIONALE section in *posix_spawn_file_actions_addclose*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawn_file_actions_addclose*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_destroy, posix_spawnattr_init — destroy and initialize spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

#include <spawn.h>

int posix_spawnattr_destroy(posix_spawnattr_t **attr*);
int posix_spawnattr_init(posix_spawnattr_t **attr*);

DESCRIPTION

The *posix_spawnattr_destroy*() function shall destroy a spawn attributes object. A destroyed *attr* attributes object can be reinitialized using *posix_spawnattr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause *posix_spawnattr_destroy*() to set the object referenced by *attr* to an invalid value.

The *posix_spawnattr_init*() function shall initialize a spawn attributes object *attr* with the default value for all of the individual attributes used by the implementation. Results are undefined if *posix_spawnattr_init*() is called specifying an already initialized *attr* attributes object.

A spawn attributes object is of type **posix_spawnattr_t** (defined in *<spawn.h>*) and is used to specify the inheritance of process attributes across a spawn operation. POSIX.1-2008 does not define comparison or assignment operators for the type **posix_spawnattr_t**.

Each implementation shall document the individual attributes it uses and their default values unless these values are defined by POSIX.1-2008. Attributes not defined by POSIX.1-2008, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The resulting spawn attributes object (possibly modified by setting individual attribute values), is used to modify the behavior of *posix_spawn*() or *posix_spawnp*(). After a spawn attributes object has been used to spawn a process by a call to a *posix_spawn*() or *posix_spawnp*(), any function affecting the attributes object (including destruction) shall not affect any process that has been spawned in this way.

RETURN VALUE

Upon successful completion, *posix_spawnattr_destroy*() and *posix_spawnattr_init*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *posix_spawnattr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the spawn attributes object.

The *posix_spawnattr_destroy*() function may fail if:

EINVAL
The value specified by attr is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

The original spawn interface proposed in POSIX.1-2008 defined the attributes that specify the inheritance of process attributes across a spawn operation as a structure. In order to be able to separate optional individual attributes under their appropriate options (that is, the *spawn-schedparam* and *spawn-schedpolicy* attributes depending upon the Process Scheduling option), and also for extensibility and consistency with the newer POSIX interfaces, the attributes interface has been changed to an opaque data type. This interface now consists of the type **posix_spawnattr_t**, representing a spawn attributes object, together with associated functions to initialize or destroy the attributes object, and to set or get each individual attribute. Although the new object-oriented interface is more verbose than the original structure, it is simple to use, more extensible, and easy to implement.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getflags, posix_spawnattr_setflags — get and set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawnattr_getflags(const posix_spawnattr_t *restrict attr,
    short *restrict flags);
int posix_spawnattr_setflags(posix_spawnattr_t *attr, short flags);
```

DESCRIPTION

The *posix_spawnattr_getflags*() function shall obtain the value of the *spawn-flags* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setflags*() function shall set the *spawn-flags* attribute in an initialized attributes object referenced by *attr*.

The *spawn-flags* attribute is used to indicate which process attributes are to be changed in the new process image when invoking *posix_spawn*() or *posix_spawnp*(). It is the bitwise-inclusive OR of zero or more of the following flags:

POSIX_SPAWN_RESETIDS
POSIX_SPAWN_SETPGROUP
POSIX_SPAWN_SETSIGDEF
POSIX_SPAWN_SETSIGMASK
POSIX_SPAWN_SETSCHEDPARAM
POSIX_SPAWN_SETSCHEDULER

These flags are defined in *<spawn.h>*. The default value of this attribute shall be as if no flags were set.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getflags*() shall return zero and store the value of the *spawn-flags* attribute of *attr* into the object referenced by the *flags* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setflags*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setflags*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getpgroup, posix_spawnattr_setpgroup — get and set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawnattr_getpgroup(const posix_spawnattr_t *restrict attr,
   pid_t *restrict pgroup);
int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

DESCRIPTION

The *posix_spawnattr_getpgroup*() function shall obtain the value of the *spawn-pgroup* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setpgroup*() function shall set the *spawn-pgroup* attribute in an initialized attributes object referenced by *attr*.

The *spawn-pgroup* attribute represents the process group to be joined by the new process image in a spawn operation (if POSIX_SPAWN_SETPGROUP is set in the *spawn-flags* attribute). The default value of this attribute shall be zero.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getpgroup*() shall return zero and store the value of the *spawn-pgroup* attribute of *attr* into the object referenced by the *pgroup* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setpgroup*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setpgroup*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getschedparam, posix_spawnattr_setschedparam — get and set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedparam(const posix_spawnattr_t
   *restrict attr, struct sched_param *restrict schedparam);
int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
   const struct sched_param *restrict schedparam);
```

DESCRIPTION

The *posix_spawnattr_getschedparam*() function shall obtain the value of the *spawn-schedparam* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedparam*() function shall set the *spawn-schedparam* attribute in an initialized attributes object referenced by *attr*.

The *spawn-schedparam* attribute represents the scheduling parameters to be assigned to the new process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER or POSIX_SPAWN_SETSCHEDPARAM is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getschedparam*() shall return zero and store the value of the *spawn-schedparam* attribute of *attr* into the object referenced by the *schedparam* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setschedparam*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setschedparam*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn and Process Scheduling options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getschedpolicy, posix_spawnattr_setschedpolicy — get and set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>
#include <sched.h>

int posix_spawnattr_getschedpolicy(const posix_spawnattr_t
   *restrict attr, int *restrict schedpolicy);
int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
   int schedpolicy);
```

DESCRIPTION

The *posix_spawnattr_getschedpolicy*() function shall obtain the value of the *spawn-schedpolicy* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setschedpolicy*() function shall set the *spawn-schedpolicy* attribute in an initialized attributes object referenced by *attr*.

The *spawn-schedpolicy* attribute represents the scheduling policy to be assigned to the new process image in a spawn operation (if POSIX_SPAWN_SETSCHEDULER is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getschedpolicy*() shall return zero and store the value of the *spawn-schedpolicy* attribute of *attr* into the object referenced by the *schedpolicy* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setschedpolicy*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setschedpolicy*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn and Process Scheduling options and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getsigdefault, posix_spawnattr_setsigdefault — get and set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigdefault(const posix_spawnattr_t
   *restrict attr, sigset_t *restrict sigdefault);
int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
   const sigset_t *restrict sigdefault);
```

DESCRIPTION

The *posix_spawnattr_getsigdefault*() function shall obtain the value of the *spawn-sigdefault* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setsigdefault*() function shall set the *spawn-sigdefault* attribute in an initialized attributes object referenced by *attr*.

The *spawn-sigdefault* attribute represents the set of signals to be forced to default signal handling in the new process image (if POSIX_SPAWN_SETSIGDEF is set in the *spawn-flags* attribute) by a spawn operation. The default value of this attribute shall be an empty signal set.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getsigdefault*() shall return zero and store the value of the *spawn-sigdefault* attribute of *attr* into the object referenced by the *sigdefault* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setsigdefault*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setsigdefault*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( ), *posix_spawnattr_getsigmask*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_getsigmask, posix_spawnattr_setsigmask — get and set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_getsigmask(const posix_spawnattr_t *restrict attr,
    sigset_t *restrict sigmask);
int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

DESCRIPTION

The *posix_spawnattr_getsigmask*() function shall obtain the value of the *spawn-sigmask* attribute from the attributes object referenced by *attr*.

The *posix_spawnattr_setsigmask*() function shall set the *spawn-sigmask* attribute in an initialized attributes object referenced by *attr*.

The *spawn-sigmask* attribute represents the signal mask in effect in the new process image of a spawn operation (if POSIX_SPAWN_SETSIGMASK is set in the *spawn-flags* attribute). The default value of this attribute is unspecified.

RETURN VALUE

Upon successful completion, *posix_spawnattr_getsigmask*() shall return zero and store the value of the *spawn-sigmask* attribute of *attr* into the object referenced by the *sigmask* parameter; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *posix_spawnattr_setsigmask*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *attr* is invalid.

The *posix_spawnattr_setsigmask*() function may fail if:

EINVAL
The value of the attribute being set is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are part of the Spawn option and need not be provided on all implementations.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

posix_spawn( ), *posix_spawnattr_destroy*( ), *posix_spawnattr_getsigdefault*( ), *posix_spawnattr_getflags*( ), *posix_spawnattr_getpgroup*( ), *posix_spawnattr_getschedparam*( ), *posix_spawnattr_getschedpolicy*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<spawn.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_init — initialize the spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

#include <spawn.h>

int posix_spawnattr_init(posix_spawnattr_t **attr*);

DESCRIPTION

Refer to *posix_spawnattr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setflags — set the spawn-flags attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

#include <spawn.h>

int posix_spawnattr_setflags(posix_spawnattr_t ***attr*, short *flags*);

DESCRIPTION

Refer to *posix_spawnattr_getflags*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setpgroup — set the spawn-pgroup attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawnattr_setpgroup(posix_spawnattr_t *attr, pid_t pgroup);
```

DESCRIPTION

Refer to *posix_spawnattr_getpgroup*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setschedparam — set the spawn-schedparam attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedparam(posix_spawnattr_t *restrict attr,
    const struct sched_param *restrict schedparam);
```

DESCRIPTION

Refer to *posix_spawnattr_getschedparam*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setschedpolicy — set the spawn-schedpolicy attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <sched.h>
#include <spawn.h>

int posix_spawnattr_setschedpolicy(posix_spawnattr_t *attr,
   int schedpolicy);
```

DESCRIPTION

Refer to *posix_spawnattr_getschedpolicy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setsigdefault — set the spawn-sigdefault attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigdefault(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigdefault);
```

DESCRIPTION

Refer to *posix_spawnattr_getsigdefault*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnattr_setsigmask — set the spawn-sigmask attribute of a spawn attributes object (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <signal.h>
#include <spawn.h>

int posix_spawnattr_setsigmask(posix_spawnattr_t *restrict attr,
    const sigset_t *restrict sigmask);
```

DESCRIPTION

Refer to *posix_spawnattr_getsigmask*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_spawnp — spawn a process (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <spawn.h>

int posix_spawnp(pid_t *restrict pid, const char *restrict file,
    const posix_spawn_file_actions_t *file_actions,
    const posix_spawnattr_t *restrict attrp,
    char *const argv[restrict], char *const envp[restrict]);
```

DESCRIPTION

Refer to *posix_spawn*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_destroy, posix_trace_attr_init — destroy and initialize the trace stream attributes object (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_destroy(trace_attr_t *attr);
int posix_trace_attr_init(trace_attr_t *attr);
```

DESCRIPTION

The *posix_trace_attr_destroy*() function shall destroy an initialized trace attributes object. A destroyed *attr* attributes object can be reinitialized using *posix_trace_attr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

The *posix_trace_attr_init*() function shall initialize a trace attributes object *attr* with the default value for all of the individual attributes used by a given implementation. The read-only *generation-version* and *clock-resolution* attributes of the newly initialized trace attributes object shall be set to their appropriate values (see *Section 2.11.1.2*, *posix_trace_status_info Structure*).

Results are undefined if *posix_trace_attr_init*() is called specifying an already initialized *attr* attributes object.

Implementations may add extensions to the trace attributes object structure as permitted in the Base Definitions volume of POSIX.1-2017, *Chapter 2*, *Conformance*.

The resulting attributes object (possibly modified by setting individual attributes values), when used by *posix_trace_create*(), defines the attributes of the trace stream created. A single attributes object can be used in multiple calls to *posix_trace_create*(). After one or more trace streams have been created using an attributes object, any function affecting that attributes object, including destruction, shall not affect any trace stream previously created. An initialized attributes object also serves to receive the attributes of an existing trace stream or trace log when calling the *posix_trace_get_attr*() function.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

ERRORS

The *posix_trace_attr_destroy*() function may fail if:

EINVAL
The value of *attr* is invalid.

The *posix_trace_attr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the trace attributes object.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_attr_destroy*() and *posix_trace_attr_init*() functions may be removed in a future version.

SEE ALSO

posix_trace_create( ), *posix_trace_get_attr*( ), *uname*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getclockres, posix_trace_attr_getcreatetime, posix_trace_attr_getgenversion, posix_trace_attr_getname, posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

SYNOPSIS

```
#include <time.h>
#include <trace.h>

int posix_trace_attr_getclockres(const trace_attr_t *attr,
    struct timespec *resolution);
int posix_trace_attr_getcreatetime(const trace_attr_t *attr,
    struct timespec *createtime);

#include <trace.h>

int posix_trace_attr_getgenversion(const trace_attr_t *attr,
    char *genversion);
int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
int posix_trace_attr_setname(trace_attr_t *attr,
    const char *tracename);
```

DESCRIPTION

The *posix_trace_attr_getclockres*() function shall copy the clock resolution of the clock used to generate timestamps from the *clock-resolution* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *resolution* argument.

The *posix_trace_attr_getcreatetime*() function shall copy the trace stream creation time from the *creation-time* attribute of the attributes object pointed to by the *attr* argument into the structure pointed to by the *createtime* argument. The *creation-time* attribute shall represent the time of creation of the trace stream.

The *posix_trace_attr_getgenversion*() function shall copy the string containing version information from the *generation-version* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *genversion* argument. The *genversion* argument shall be the address of a character array which can store at least {TRACE_NAME_MAX} characters.

The *posix_trace_attr_getname*() function shall copy the string containing the trace name from the *trace-name* attribute of the attributes object pointed to by the *attr* argument into the string pointed to by the *tracename* argument. The *tracename* argument shall be the address of a character array which can store at least {TRACE_NAME_MAX} characters.

The *posix_trace_attr_setname*() function shall set the name in the *trace-name* attribute of the attributes object pointed to by the *attr* argument, using the trace name string supplied by the *tracename* argument. If the supplied string contains more than {TRACE_NAME_MAX} characters, the name copied into the *trace-name* attribute may be truncated to one less than the length of {TRACE_NAME_MAX} characters. The default value is a null string.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_attr_getclockres*() function stores the *clock-resolution* attribute value in the object pointed to by *resolution*. Otherwise, the content of this object is unspecified.

If successful, the *posix_trace_attr_getcreatetime*() function stores the trace stream creation time in the object pointed to by *createtime*. Otherwise, the content of this object is unspecified.

If successful, the *posix_trace_attr_getgenversion*() function stores the trace version information in the string pointed to by *genversion*. Otherwise, the content of this string is unspecified.

If successful, the *posix_trace_attr_getname*() function stores the trace name in the string pointed to by *tracename*. Otherwise, the content of this string is unspecified.

ERRORS

The *posix_trace_attr_getclockres*(), *posix_trace_attr_getcreatetime*(), *posix_trace_attr_getgenversion*(), and *posix_trace_attr_getname*() functions may fail if:

EINVAL
The value specified by one of the arguments is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_attr_getclockres*(), *posix_trace_attr_getcreatetime*(), *posix_trace_attr_getgenversion*(), *posix_trace_attr_getname*(), and *posix_trace_attr_setname*() functions may be removed in a future version.

SEE ALSO

posix_trace_attr_destroy( ), *posix_trace_create*( ), *posix_trace_get_attr*( ), *uname*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getinherited, posix_trace_attr_getlogfullpolicy, posix_trace_attr_getstreamfullpolicy, posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy, posix_trace_attr_setstreamfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_getinherited(const trace_attr_t *restrict attr,
    int *restrict inheritancepolicy);
int posix_trace_attr_getlogfullpolicy(const trace_attr_t *restrict attr,
    int *restrict logpolicy);
int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
    attr, int *restrict streampolicy);
int posix_trace_attr_setinherited(trace_attr_t *attr,
    int inheritancepolicy);
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
    int logpolicy);
int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

DESCRIPTION

The *posix_trace_attr_getinherited*() and *posix_trace_attr_setinherited*() functions, respectively, shall get and set the inheritance policy stored in the *inheritance* attribute for traced processes across the *fork*() and *spawn*() operations. The *inheritance* attribute of the attributes object pointed to by the *attr* argument shall be set to one of the following values defined by manifest constants in the *<trace.h>* header:

POSIX_TRACE_CLOSE_FOR_CHILD
: After a *fork*() or *spawn*() operation, the child shall not be traced, and tracing of the parent shall continue.

POSIX_TRACE_INHERITED
: After a *fork*() or *spawn*() operation, if the parent is being traced, its child shall be concurrently traced using the same trace stream.

The default value for the *inheritance* attribute is POSIX_TRACE_CLOSE_FOR_CHILD.

The *posix_trace_attr_getlogfullpolicy*() and *posix_trace_attr_setlogfullpolicy*() functions, respectively, shall get and set the trace log full policy stored in the *log-full-policy* attribute of the attributes object pointed to by the *attr* argument.

The *log-full-policy* attribute shall be set to one of the following values defined by manifest constants in the *<trace.h>* header:

POSIX_TRACE_LOOP
: The trace log shall loop until the associated trace stream is stopped. This policy means that when the trace log gets full, the file system shall reuse the resources allocated to the oldest trace events that were recorded. In this way, the trace log will always contain the most recent trace events flushed.

POSIX_TRACE_UNTIL_FULL
: The trace stream shall be flushed to the trace log until the trace log is full. This condition can be deduced from the *posix_log_full_status* member status (see the **posix_trace_status_info** structure defined in *<trace.h>*). The last recorded trace event shall be the POSIX_TRACE_STOP trace event.

POSIX_TRACE_APPEND
: The associated trace stream shall be flushed to the trace log without log size limitation. If the application specifies POSIX_TRACE_APPEND, the implementation shall ignore the *log-max-size* attribute.

The default value for the *log-full-policy* attribute is POSIX_TRACE_LOOP.

The *posix_trace_attr_getstreamfullpolicy*() and *posix_trace_attr_setstreamfullpolicy*() functions, respectively, shall get and set the trace stream full policy stored in the *stream-full-policy* attribute of the attributes object pointed to by the *attr* argument.

The *stream-full-policy* attribute shall be set to one of the following values defined by manifest constants in the <*trace.h*> header:

POSIX_TRACE_LOOP
: The trace stream shall loop until explicitly stopped by the *posix_trace_stop*() function. This policy means that when the trace stream is full, the trace system shall reuse the resources allocated to the oldest trace events recorded. In this way, the trace stream will always contain the most recent trace events recorded.

POSIX_TRACE_UNTIL_FULL
: The trace stream will run until the trace stream resources are exhausted. Then the trace stream will stop. This condition can be deduced from *posix_stream_status* and *posix_stream_full_status* (see the **posix_trace_status_info** structure defined in <*trace.h*>). When this trace stream is read, a POSIX_TRACE_STOP trace event shall be reported after reporting the last recorded trace event. The trace system shall reuse the resources allocated to any trace events already reported—see the *posix_trace_getnext_event*(), *posix_trace_trygetnext_event*(), and *posix_trace_timedgetnext_event*() functions—or already flushed for an active trace stream with log if the Trace Log option is supported; see the *posix_trace_flush*() function. The trace system shall restart the trace stream when it is empty and may restart it sooner. A POSIX_TRACE_START trace event shall be reported before reporting the next recorded trace event.

POSIX_TRACE_FLUSH
: If the Trace Log option is supported, this policy is identical to the POSIX_TRACE_UNTIL_FULL trace stream full policy except that the trace stream shall be flushed regularly as if *posix_trace_flush*() had been explicitly called. Defining this policy for an active trace stream without log shall be invalid.

The default value for the *stream-full-policy* attribute shall be POSIX_TRACE_LOOP for an active trace stream without log.

If the Trace Log option is supported, the default value for the *stream-full-policy* attribute shall be POSIX_TRACE_FLUSH for an active trace stream with log.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_attr_getinherited*() function shall store the *inheritance* attribute value in the object pointed to by *inheritancepolicy*. Otherwise, the content of this object is undefined.

If successful, the *posix_trace_attr_getlogfullpolicy*() function shall store the *log-full-policy* attribute value in the object pointed to by *logpolicy*. Otherwise, the content of this object is undefined.

If successful, the *posix_trace_attr_getstreamfullpolicy*() function shall store the *stream-full-policy* attribute value in the object pointed to by *streampolicy*. Otherwise, the content of this object is undefined.

ERRORS

These functions may fail if:

EINVAL
: The value specified by at least one of the arguments is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The following functions:

> *posix_trace_attr_getinherited*()
> *posix_trace_attr_getlogfullpolicy*()
> *posix_trace_attr_getstreamfullpolicy*()
> *posix_trace_attr_setinherited*()
> *posix_trace_attr_setlogfullpolicy*()
> *posix_trace_attr_setstreamfullpolicy*()

may be removed in a future version.

SEE ALSO

fork( ), *posix_trace_attr_destroy*( ), *posix_trace_create*( ), *posix_trace_get_attr*( ), *posix_trace_getnext_event*( ), *posix_trace_start*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getlogsize, posix_trace_attr_getmaxdatasize, posix_trace_attr_getmaxsystemeventsize, posix_trace_attr_getmaxusereventsize, posix_trace_attr_getstreamsize, posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize, posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_getlogsize(const trace_attr_t *restrict attr,
   size_t *restrict logsize);
int posix_trace_attr_getmaxdatasize(const trace_attr_t *restrict attr,
   size_t *restrict maxdatasize);
int posix_trace_attr_getmaxsystemeventsize(
   const trace_attr_t *restrict attr,
   size_t *restrict eventsize);
int posix_trace_attr_getmaxusereventsize(
   const trace_attr_t *restrict attr,
   size_t data_len, size_t *restrict eventsize);
int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
   size_t *restrict streamsize);
int posix_trace_attr_setlogsize(trace_attr_t *attr,
   size_t logsize);
int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
   size_t maxdatasize);
int posix_trace_attr_setstreamsize(trace_attr_t *attr,
   size_t streamsize);
```

DESCRIPTION

The *posix_trace_attr_getlogsize*() function shall copy the log size, in bytes, from the *log-max-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *logsize* argument. This log size is the maximum total of bytes that shall be allocated for system and user trace events in the trace log. The default value for the *log-max-size* attribute is implementation-defined.

The *posix_trace_attr_setlogsize*() function shall set the maximum allowed size, in bytes, in the *log-max-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *logsize* argument.

The trace log size shall be used if the *log-full-policy* attribute is set to POSIX_TRACE_LOOP or POSIX_TRACE_UNTIL_FULL. If the *log-full-policy* attribute is set to POSIX_TRACE_APPEND, the implementation shall ignore the *log-max-size* attribute.

The *posix_trace_attr_getmaxdatasize*() function shall copy the maximum user trace event data size, in bytes, from the *max-data-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *maxdatasize* argument. The default value for the *max-data-size* attribute is implementation-defined.

The *posix_trace_attr_getmaxsystemeventsize*() function shall calculate the maximum memory size, in bytes, required to store a single system trace event. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

The values returned as the maximum memory sizes of the user and system trace events shall be such that if the sum of the maximum memory sizes of a set of the trace events that may be recorded in a trace stream is less than or equal to the *stream-min-size* attribute of that trace stream, the system provides the necessary resources for recording all those trace events, without loss.

The *posix_trace_attr_getmaxusereventsize*() function shall calculate the maximum memory size, in bytes, required to store a single user trace event generated by a call to *posix_trace_event*() with a *data_len* parameter equal to the *data_len* value specified in this call. This value is calculated for the trace stream attributes object pointed to by the *attr* argument and is returned in the variable pointed to by the *eventsize* argument.

The *posix_trace_attr_getstreamsize*() function shall copy the stream size, in bytes, from the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument into the variable pointed to by the *streamsize* argument.

This stream size is the current total memory size reserved for system and user trace events in the trace stream. The default value for the *stream-min-size* attribute is implementation-defined. The stream size refers to memory used to store trace event records. Other stream data (for example, trace attribute values) shall not be included in this size.

The *posix_trace_attr_setmaxdatasize*() function shall set the maximum allowed size, in bytes, in the *max-data-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *maxdatasize* argument. This maximum size is the maximum allowed size for the user data argument which may be passed to *posix_trace_event*(). The implementation shall be allowed to truncate data passed to *trace_user_event* which is longer than *maxdatasize*.

The *posix_trace_attr_setstreamsize*() function shall set the minimum allowed size, in bytes, in the *stream-min-size* attribute of the attributes object pointed to by the *attr* argument, using the size value supplied by the *streamsize* argument.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_attr_getlogsize*() function stores the maximum trace log allowed size in the object pointed to by *logsize*, if successful.

The *posix_trace_attr_getmaxdatasize*() function stores the maximum trace event record memory size in the object pointed to by *maxdatasize*, if successful.

The *posix_trace_attr_getmaxsystemeventsize*() function stores the maximum memory size to store a single system trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getmaxusereventsize*() function stores the maximum memory size to store a single user trace event in the object pointed to by *eventsize*, if successful.

The *posix_trace_attr_getstreamsize*() function stores the maximum trace stream allowed size in the object pointed to by *streamsize*, if successful.

ERRORS

These functions may fail if:

EINVAL
The value specified by one of the arguments is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The following functions:

posix_trace_attr_getlogsize()
posix_trace_attr_getmaxdatasize()
posix_trace_attr_getmaxsystemeventsize()
posix_trace_attr_getmaxusereventsize()
posix_trace_attr_getstreamsize()
posix_trace_attr_setlogsize()
posix_trace_attr_setmaxdatasize()
posix_trace_attr_setstreamsize()

may be removed in a future version.

SEE ALSO

posix_trace_attr_destroy( ), *posix_trace_create*( ), *posix_trace_event*( ), *posix_trace_get_attr*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getname — retrieve and set information about a trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_getname(const trace_attr_t *attr,
    char *tracename);
```

DESCRIPTION

Refer to *posix_trace_attr_getclockres*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getstreamfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

#include <trace.h>

int posix_trace_attr_getstreamfullpolicy(const trace_attr_t *restrict
   attr, int *restrict *streampolicy*);

DESCRIPTION

Refer to *posix_trace_attr_getinherited*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_getstreamsize — retrieve and set trace stream size attributes (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_getstreamsize(const trace_attr_t *restrict attr,
   size_t *restrict streamsize);
```

DESCRIPTION

Refer to *posix_trace_attr_getlogsize*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_init — initialize the trace stream attributes object (**TRACING**)

SYNOPSIS

#include <trace.h>

int posix_trace_attr_init(trace_attr_t **attr*);

DESCRIPTION

Refer to *posix_trace_attr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_setinherited, posix_trace_attr_setlogfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_setinherited(trace_attr_t *attr,
   int inheritancepolicy);
int posix_trace_attr_setlogfullpolicy(trace_attr_t *attr,
   int logpolicy);
```

DESCRIPTION

Refer to *posix_trace_attr_getinherited*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_setlogsize, posix_trace_attr_setmaxdatasize — retrieve and set trace stream size attributes (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_setlogsize(trace_attr_t *attr,
   size_t logsize);
int posix_trace_attr_setmaxdatasize(trace_attr_t *attr,
   size_t maxdatasize);
```

DESCRIPTION

Refer to *posix_trace_attr_getlogsize*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_setname — retrieve and set information about a trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_setname(trace_attr_t *attr,
   const char *tracename);
```

DESCRIPTION

Refer to *posix_trace_attr_getclockres*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_setstreamfullpolicy — retrieve and set the behavior of a trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_attr_setstreamfullpolicy(trace_attr_t *attr,
    int streampolicy);
```

DESCRIPTION

Refer to *posix_trace_attr_getinherited*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_attr_setstreamsize — retrieve and set trace stream size attributes (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_attr_setstreamsize(trace_attr_t *attr,
   size_t streamsize);
```

DESCRIPTION

Refer to *posix_trace_attr_getlogsize*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_clear — clear trace stream and trace log (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_clear(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_clear*() function shall reinitialize the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create*() function, except that the same allocated resources shall be reused, the mapping of trace event type identifiers to trace event names shall be unchanged, and the trace stream status shall remain unchanged (that is, if it was running, it remains running and if it was suspended, it remains suspended).

All trace events in the trace stream recorded before the call to *posix_trace_clear*() shall be lost. The *posix_stream_full_status* status shall be set to POSIX_TRACE_NOT_FULL. There is no guarantee that all trace events that occurred during the *posix_trace_clear*() call are recorded; the behavior with respect to trace points that may occur during this call is unspecified.

If the Trace Log option is supported and the trace stream has been created with a log, the *posix_trace_clear*() function shall reinitialize the trace stream with the same behavior as if the trace stream was created without the log, plus it shall reinitialize the trace log associated with the trace stream identified by the argument *trid* as if it were returning from the *posix_trace_create_withlog*() function, except that the same allocated resources, for the trace log, may be reused and the associated trace stream status remains unchanged. The first trace event recorded in the trace log after the call to *posix_trace_clear*() shall be the same as the first trace event recorded in the active trace stream after the call to *posix_trace_clear*(). The *posix_log_full_status* status shall be set to POSIX_TRACE_NOT_FULL. There is no guarantee that all trace events that occurred during the *posix_trace_clear*() call are recorded in the trace log; the behavior with respect to trace points that may occur during this call is unspecified. If the log full policy is POSIX_TRACE_APPEND, the effect of a call to this function is unspecified for the trace log associated with the trace stream identified by the *trid* argument.

RETURN VALUE

Upon successful completion, the *posix_trace_clear*() function shall return a value of zero. Otherwise, it shall return the corresponding error number.

ERRORS

The *posix_trace_clear*() function shall fail if:

EINVAL
The value of the *trid* argument does not correspond to an active trace stream.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_clear*() function may be removed in a future version.

SEE ALSO

posix_trace_attr_destroy( ), *posix_trace_create*( ), *posix_trace_get_attr*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_close, posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)

SYNOPSIS

#include <trace.h>

int posix_trace_close(trace_id_t *trid*);
int posix_trace_open(int *file_desc*, trace_id_t **trid*);
int posix_trace_rewind(trace_id_t *trid*);

DESCRIPTION

The *posix_trace_close*() function shall deallocate the trace log identifier indicated by *trid*, and all of its associated resources. If there is no valid trace log pointed to by the *trid*, this function shall fail.

The *posix_trace_open*() function shall allocate the necessary resources and establish the connection between a trace log identified by the *file_desc* argument and a trace stream identifier identified by the object pointed to by the *trid* argument. The *file_desc* argument should be a valid open file descriptor that corresponds to a trace log. The *file_desc* argument shall be open for reading. The current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event*(), shall be set to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

The *posix_trace_open*() function shall return a trace stream identifier in the variable pointed to by the *trid* argument, that may only be used by the following functions:

tab(!); l l. T{
posix_trace_close()
posix_trace_eventid_equal()
posix_trace_eventid_get_name()
posix_trace_eventtypelist_getnext_id()
posix_trace_eventtypelist_rewind()
T}!T{
posix_trace_get_attr()
posix_trace_get_status()
posix_trace_getnext_event()
posix_trace_rewind()
T}

In particular, notice that the operations normally used by a trace controller process, such as *posix_trace_start*(), *posix_trace_stop*(), or *posix_trace_shutdown*(), cannot be invoked using the trace stream identifier returned by the *posix_trace_open*() function.

The *posix_trace_rewind*() function shall reset the current trace event timestamp, which specifies the timestamp of the trace event that will be read by the next call to *posix_trace_getnext_event*(), to the timestamp of the oldest trace event recorded in the trace log identified by *trid*.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, the *posix_trace_open*() function stores the trace stream identifier value in the object pointed to by *trid*.

ERRORS

The *posix_trace_open*() function shall fail if:

EINTR
The operation was interrupted by a signal and thus no trace log was opened.

EINVAL
The object pointed to by *file_desc* does not correspond to a valid trace log.

The *posix_trace_close*() and *posix_trace_rewind*() functions may fail if:

EINVAL
The object pointed to by *trid* does not correspond to a valid trace log.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_close*(), *posix_trace_open*(), and *posix_trace_rewind*() functions may be removed in a future version.

SEE ALSO

posix_trace_get_attr( ), *posix_trace_get_filter*( ), *posix_trace_getnext_event*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_create, posix_trace_create_withlog, posix_trace_flush, posix_trace_shutdown — trace stream initialization, flush, and shutdown from a process (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_create(pid_t pid,
   const trace_attr_t *restrict attr,
   trace_id_t *restrict trid);
int posix_trace_create_withlog(pid_t pid,
   const trace_attr_t *restrict attr, int file_desc,
   trace_id_t *restrict trid);
int posix_trace_flush(trace_id_t trid);
int posix_trace_shutdown(trace_id_t trid);
```

DESCRIPTION

The *posix_trace_create*() function shall create an active trace stream. It allocates all the resources needed by the trace stream being created for tracing the process specified by *pid* in accordance with the *attr* argument. The *attr* argument represents the initial attributes of the trace stream and shall have been initialized by the function *posix_trace_attr_init*() prior to the *posix_trace_create*() call. If the argument *attr* is NULL, the default attributes shall be used. The *attr* attributes object shall be manipulated through a set of functions described in the *posix_trace_attr* family of functions. If the attributes of the object pointed to by *attr* are modified later, the attributes of the trace stream shall not be affected. The *creation-time* attribute of the newly created trace stream shall be set to the value of the system clock, if the Timers option is not supported, or to the value of the CLOCK_REALTIME clock, if the Timers option is supported.

The *pid* argument represents the target process to be traced. If the process executing this function does not have appropriate privileges to trace the process identified by *pid*, an error shall be returned. If the *pid* argument is zero, the calling process shall be traced.

The *posix_trace_create*() function shall store the trace stream identifier of the new trace stream in the object pointed to by the *trid* argument. This trace stream identifier shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:

tab(!); l l. T{
posix_trace_clear()
posix_trace_eventid_equal()
posix_trace_eventid_get_name()
posix_trace_eventtypelist_getnext_id()
posix_trace_eventtypelist_rewind()
posix_trace_get_attr()
posix_trace_get_status()
T}!T{
posix_trace_getnext_event()
posix_trace_shutdown()
posix_trace_start()
posix_trace_stop()
posix_trace_timedgetnext_event()
posix_trace_trid_eventid_open()
posix_trace_trygetnext_event()
T}

If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:

tab(!); l l. T{ *posix_trace_get_filter*() T}!T{ *posix_trace_set_filter*() T}

In particular, notice that the operations normally used by a trace analyzer process, such as *posix_trace_rewind*() or *posix_trace_close*(), cannot be invoked using the trace stream identifier returned by the *posix_trace_create*() function.

A trace stream shall be created in a suspended state. If the Trace Event Filter option is supported, its trace event type filter shall be empty.

The *posix_trace_create*() function may be called multiple times from the same or different processes, with the system-wide limit indicated by the runtime invariant value {TRACE_SYS_MAX}, which has the minimum value {_POSIX_TRACE_SYS_MAX}.

The trace stream identifier returned by the *posix_trace_create*() function in the argument pointed to by *trid* is valid only in the process that made the function call. If it is used from another process, that is a child process, in functions defined in POSIX.1-2008, these functions shall return with the error **[EINVAL]**.

The *posix_trace_create_withlog*() function shall be equivalent to *posix_trace_create*(), except that it associates a trace log with this stream. The *file_desc* argument shall be the file descriptor designating the trace log destination. The function shall fail if this file descriptor refers to a file with a file type that is not compatible with the log policy associated with the trace log. The list of the appropriate file types that are compatible with each log policy is implementation-defined.

The *posix_trace_create_withlog*() function shall return in the parameter pointed to by *trid* the trace stream identifier, which uniquely identifies the newly created trace stream, and shall be used in subsequent calls to control tracing. The *trid* argument may only be used by the following functions:

tab(!); l l. T{
posix_trace_clear()
posix_trace_eventid_equal()
posix_trace_eventid_get_name()
posix_trace_eventtypelist_getnext_id()
posix_trace_eventtypelist_rewind()
posix_trace_flush()
posix_trace_get_attr()
T}!T{
posix_trace_get_status()
posix_trace_getnext_event()
posix_trace_shutdown()
posix_trace_start()
posix_trace_stop()
posix_trace_timedgetnext_event()
posix_trace_trid_eventid_open()
T}

If the Trace Event Filter option is supported, the following additional functions may use the *trid* argument:

tab(!); l l. T{ *posix_trace_get_filter*() T}!T{ *posix_trace_set_filter*() T}

In particular, notice that the operations normally used by a trace analyzer process, such as *posix_trace_rewind*() or *posix_trace_close*(), cannot be invoked using the trace stream identifier returned by the *posix_trace_create_withlog*() function.

The *posix_trace_flush*() function shall initiate a flush operation which copies the contents of the trace stream identified by the argument *trid* into the trace log associated with the trace stream at the creation time. If no trace log has been associated with the trace stream pointed to by *trid*, this function shall return an error. The termination of the flush operation can be polled by the *posix_trace_get_status*() function. During the flush operation, it shall be possible to trace new trace events up to the point when the trace stream becomes full. After flushing is completed, the space used by the flushed trace events shall be available for tracing new trace events.

If flushing the trace stream causes the resulting trace log to become full, the trace log full policy shall be applied. If the trace *log-full-policy* attribute is set, the following occurs:

POSIX_TRACE_UNTIL_FULL
: The trace events that have not yet been flushed shall be discarded.

POSIX_TRACE_LOOP
: The trace events that have not yet been flushed shall be written to the beginning of the trace log, overwriting previous trace events stored there.

POSIX_TRACE_APPEND
: The trace events that have not yet been flushed shall be appended to the trace log.

The *posix_trace_shutdown*() function shall stop the tracing of trace events in the trace stream identified by *trid*, as if *posix_trace_stop*() had been invoked. The *posix_trace_shutdown*() function shall free all the resources associated with the trace stream.

The *posix_trace_shutdown*() function shall not return until all the resources associated with the trace stream have been freed. When the *posix_trace_shutdown*() function returns, the *trid* argument becomes an invalid trace stream identifier. A call to this function shall unconditionally deallocate the resources regardless of whether all trace events have been retrieved by the analyzer process. Any thread blocked on one of the *trace_getnext_event*() functions (which specified this *trid*) before this call is unblocked with the error **[EINVAL]**.

If the process exits, invokes a member of the *exec* family of functions, or is terminated, the trace streams that the process had created and that have not yet been shut down, shall be automatically shut down as if an explicit call were made to the *posix_trace_shutdown*() function.

For an active trace stream with log, when the *posix_trace_shutdown*() function is called, all trace events that have not yet been flushed to the trace log shall be flushed, as in the *posix_trace_flush*() function, and the trace log shall be closed.

When a trace log is closed, all the information that may be retrieved later from the trace log through the trace interface shall have been written to the trace log. This information includes the trace attributes, the list of trace event types (with the mapping between trace event names and trace event type identifiers), and the trace status.

In addition, unspecified information shall be written to the trace log to allow detection of a valid trace log during the *posix_trace_open*() operation.

The *posix_trace_shutdown*() function shall not return until all trace events have been flushed.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_create*() and *posix_trace_create_withlog*() functions store the trace stream identifier value in the object pointed to by *trid*, if successful.

ERRORS

The *posix_trace_create*() and *posix_trace_create_withlog*() functions shall fail if:

EAGAIN
: No more trace streams can be started now. {TRACE_SYS_MAX} has been exceeded.

EINTR
: The operation was interrupted by a signal. No trace stream was created.

EINVAL
: One or more of the trace parameters specified by the *attr* parameter is invalid.

ENOMEM
: The implementation does not currently have sufficient memory to create the trace stream with the specified parameters.

EPERM
The caller does not have appropriate privileges to trace the process specified by *pid*.

ESRCH
The *pid* argument does not refer to an existing process.

The *posix_trace_create_withlog*() function shall fail if:

EBADF
The *file_desc* argument is not a valid file descriptor open for writing.

EINVAL
The *file_desc* argument refers to a file with a file type that does not support the log policy associated with the trace log.

ENOSPC
No space left on device. The device corresponding to the argument *file_desc* does not contain the space required to create this trace log.

The *posix_trace_flush*() and *posix_trace_shutdown*() functions shall fail if:

EINVAL
The value of the *trid* argument does not correspond to an active trace stream with log.

EFBIG
The trace log file has attempted to exceed an implementation-defined maximum file size.

ENOSPC
No space left on device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_create*(), *posix_trace_create_withlog*(), *posix_trace_flush*(), and *posix_trace_shutdown*() functions may be removed in a future version.

SEE ALSO

clock_getres( ), *exec*, *posix_trace_attr_destroy*( ), *posix_trace_clear*( ), *posix_trace_close*( ), *posix_trace_eventid_equal*( ), *posix_trace_eventtypelist_getnext_id*( ), *posix_trace_get_attr*( ), *posix_trace_get_filter*( ), *posix_trace_getnext_event*( ), *posix_trace_start*( ), *posix_trace_start*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_event, posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

void posix_trace_event(trace_event_id_t event_id,
   const void *restrict data_ptr, size_t data_len);
int posix_trace_eventid_open(const char *restrict event_name,
   trace_event_id_t *restrict event_id);
```

DESCRIPTION

The *posix_trace_event*() function shall record the *event_id* and the user data pointed to by *data_ptr* in the trace stream into which the calling process is being traced and in which *event_id* is not filtered out. If the total size of the user trace event data represented by *data_len* is not greater than the declared maximum size for user trace event data, then the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_NOT_TRUNCATED. Otherwise, the user trace event data is truncated to this declared maximum size and the *truncation-status* attribute of the trace event recorded is POSIX_TRACE_TRUNCATED_RECORD.

If there is no trace stream created for the process or if the created trace stream is not running, or if the trace event specified by *event_id* is filtered out in the trace stream, the *posix_trace_event*() function shall have no effect.

The *posix_trace_eventid_open*() function shall associate a user trace event name with a trace event type identifier for the calling process. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value {TRACE_USER_EVENT_MAX}, which has the minimum value {POSIX_TRACE_USER_EVENT_MAX}.

If the Trace Inherit option is not supported, the *posix_trace_eventid_open*() function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the traced process, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see *Table 2-7*, *Trace Option: User Trace Event*) user trace event shall be returned.

If the Trace Inherit option is supported, the *posix_trace_eventid_open*() function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in this same trace stream, and is returned in the variable pointed to by the *event_id* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (*Table 2-7*, *Trace Option: User Trace Event*) user trace event shall be returned.

Note: The above procedure, together with the fact that multiple processes can only be traced into the same trace stream by inheritance, ensure that all the processes that are traced into a trace stream have the same mapping of trace event names to trace event type identifiers.

If there is no trace stream created, the *posix_trace_eventid_open*() function shall store this information for future trace streams created for this process.

RETURN VALUE

No return value is defined for the *posix_trace_event*() function.

Upon successful completion, the *posix_trace_eventid_open*() function shall return a value of zero. Otherwise, it shall return the corresponding error number. The *posix_trace_eventid_open*() function stores the trace event type identifier value in the object pointed to by *event_id*, if successful.

ERRORS

The *posix_trace_eventid_open*() function shall fail if:

ENAMETOOLONG
The size of the name pointed to by the *event_name* argument was longer than the implementation-defined value {TRACE_EVENT_NAME_MAX}.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_event*() and *posix_trace_eventid_open*() functions may be removed in a future version.

SEE ALSO

Table 2-7, *Trace Option: User Trace Event*, *exec*, *posix_trace_eventid_equal*( ), *posix_trace_start*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_eventid_equal, posix_trace_eventid_get_name, posix_trace_trid_eventid_open — manipulate the trace event type identifier (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_eventid_equal(trace_id_t trid, trace_event_id_t event1,
   trace_event_id_t event2);
int posix_trace_eventid_get_name(trace_id_t trid,
   trace_event_id_t event, char *event_name);
int posix_trace_trid_eventid_open(trace_id_t trid,
   const char *restrict event_name,
   trace_event_id_t *restrict event);
```

DESCRIPTION

The *posix_trace_eventid_equal*() function shall compare the trace event type identifiers *event1* and *event2* from the same trace stream or the same trace log identified by the *trid* argument. If the trace event type identifiers *event1* and *event2* are from different trace streams, the return value shall be unspecified.

The *posix_trace_eventid_get_name*() function shall return, in the argument pointed to by *event_name*, the trace event name associated with the trace event type identifier identified by the argument *event*, for the trace stream or for the trace log identified by the *trid* argument. The name of the trace event shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). Successive calls to this function with the same trace event type identifier and the same trace stream identifier shall return the same event name.

The *posix_trace_trid_eventid_open*() function shall associate a user trace event name with a trace event type identifier for a given trace stream. The trace stream is identified by the *trid* argument, and it shall be an active trace stream. The trace event name is the string pointed to by the argument *event_name*. It shall have a maximum of {TRACE_EVENT_NAME_MAX} characters (which has the minimum value {_POSIX_TRACE_EVENT_NAME_MAX}). The number of user trace event type identifiers that can be defined for any given process is limited by the maximum value {TRACE_USER_EVENT_MAX}, which has the minimum value {_POSIX_TRACE_USER_EVENT_MAX}.

If the Trace Inherit option is not supported, the *posix_trace_trid_eventid_open*() function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for the process being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced process, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see *Table 2-7*, *Trace Option: User Trace Event*) user trace event shall be returned.

If the Trace Inherit option is supported, the *posix_trace_trid_eventid_open*() function shall associate the user trace event name pointed to by the *event_name* argument with a trace event type identifier that is unique for all the processes being traced in the trace stream identified by the *trid* argument, and is returned in the variable pointed to by the *event* argument. If the user trace event name has already been mapped for the traced processes, then the previously assigned trace event type identifier shall be returned. If the per-process user trace event name limit represented by {TRACE_USER_EVENT_MAX} has been reached, the pre-defined POSIX_TRACE_UNNAMED_USEREVENT (see *Table 2-7*, *Trace Option: User Trace Event*) user trace event shall be returned.

RETURN VALUE

Upon successful completion, the *posix_trace_eventid_get_name*() and *posix_trace_trid_eventid_open*() functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_eventid_equal*() function shall return a non-zero value if *event1* and *event2* are equal; otherwise, a value of zero shall be returned. No errors are defined. If either *event1* or *event2* are not valid trace event type identifiers for the trace stream specified by *trid* or if the *trid* is invalid, the behavior shall be unspecified.

The *posix_trace_eventid_get_name*() function stores the trace event name value in the object pointed to by *event_name*, if successful.

The *posix_trace_trid_eventid_open*() function stores the trace event type identifier value in the object pointed to by *event*, if successful.

ERRORS

The *posix_trace_eventid_get_name*() and *posix_trace_trid_eventid_open*() functions shall fail if:

EINVAL
The *trid* argument was not a valid trace stream identifier.

The *posix_trace_trid_eventid_open*() function shall fail if:

ENAMETOOLONG
The size of the name pointed to by the *event_name* argument was longer than the implementation-defined value {TRACE_EVENT_NAME_MAX}.

The *posix_trace_eventid_get_name*() function shall fail if:

EINVAL
The trace event type identifier *event* was not associated with any name.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_eventid_equal*(), *posix_trace_eventid_get_name*(), and *posix_trace_trid_eventid_open*() functions may be removed in a future version.

SEE ALSO

Table 2-7, *Trace Option: User Trace Event*, *exec*, *posix_trace_event*( ), *posix_trace_getnext_event*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_eventid_open — trace functions for instrumenting application code (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_eventid_open(const char *restrict event_name,
   trace_event_id_t *restrict event_id);
```

DESCRIPTION

Refer to *posix_trace_event*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_eventset_add, posix_trace_eventset_del, posix_trace_eventset_empty, posix_trace_eventset_fill, posix_trace_eventset_ismember — manipulate trace event type sets (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_eventset_add(trace_event_id_t event_id,
   trace_event_set_t *set);
int posix_trace_eventset_del(trace_event_id_t event_id,
   trace_event_set_t *set);
int posix_trace_eventset_empty(trace_event_set_t *set);
int posix_trace_eventset_fill(trace_event_set_t *set, int what);
int posix_trace_eventset_ismember(trace_event_id_t event_id,
   const trace_event_set_t *restrict set, int *restrict ismember);
```

DESCRIPTION

These primitives manipulate sets of trace event types. They operate on data objects addressable by the application, not on the current trace event filter of any trace stream.

The *posix_trace_eventset_add*() and *posix_trace_eventset_del*() functions, respectively, shall add or delete the individual trace event type specified by the value of the argument *event_id* to or from the trace event type set pointed to by the argument *set*. Adding a trace event type already in the set or deleting a trace event type not in the set shall not be considered an error.

The *posix_trace_eventset_empty*() function shall initialize the trace event type set pointed to by the *set* argument such that all trace event types defined, both system and user, shall be excluded from the set.

The *posix_trace_eventset_fill*() function shall initialize the trace event type set pointed to by the argument *set*, such that the set of trace event types defined by the argument *what* shall be included in the set. The value of the argument *what* shall consist of one of the following values, as defined in the *<trace.h>* header:

POSIX_TRACE_WOPID_EVENTS
: All the process-independent implementation-defined system trace event types are included in the set.

POSIX_TRACE_SYSTEM_EVENTS
: All the implementation-defined system trace event types are included in the set, as are those defined in POSIX.1-2008.

POSIX_TRACE_ALL_EVENTS
: All trace event types defined, both system and user, are included in the set.

Applications shall call either *posix_trace_eventset_empty*() or *posix_trace_eventset_fill*() at least once for each object of type **trace_event_set_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of the *posix_trace_eventset_add*(), *posix_trace_eventset_del*(), or *posix_trace_eventset_ismember*() functions, the results are undefined.

The *posix_trace_eventset_ismember*() function shall test whether the trace event type specified by the value of the argument *event_id* is a member of the set pointed to by the argument *set*. The value returned in the object pointed to by *ismember* argument is zero if the trace event type identifier is not a member of the set and a value different from zero if it is a member of the set.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

ERRORS

These functions may fail if:

EINVAL

The value of one of the arguments is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_eventset_add*(), *posix_trace_eventset_del*(), *posix_trace_eventset_empty*(), *posix_trace_eventset_fill*(), and *posix_trace_eventset_ismember*() functions may be removed in a future version.

SEE ALSO

posix_trace_eventid_equal( ), *posix_trace_get_filter*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_eventtypelist_getnext_id, posix_trace_eventtypelist_rewind — iterate over a mapping of trace event types (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_eventtypelist_getnext_id(trace_id_t trid,
    trace_event_id_t *restrict event, int *restrict unavailable);
int posix_trace_eventtypelist_rewind(trace_id_t trid);
```

DESCRIPTION

The first time *posix_trace_eventtypelist_getnext_id*() is called, the function shall return in the variable pointed to by *event* the first trace event type identifier of the list of trace events of the trace stream identified by the *trid* argument. Successive calls to *posix_trace_eventtypelist_getnext_id*() return in the variable pointed to by *event* the next trace event type identifier in that same list. Each time a trace event type identifier is successfully written into the variable pointed to by the *event* argument, the variable pointed to by the *unavailable* argument shall be set to zero. When no more trace event type identifiers are available, and so none is returned, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The *posix_trace_eventtypelist_rewind*() function shall reset the next trace event type identifier to be read to the first trace event type identifier from the list of trace events used in the trace stream identified by *trid*.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_eventtypelist_getnext_id*() function stores the trace event type identifier value in the object pointed to by *event*, if successful.

ERRORS

These functions shall fail if:

EINVAL
The *trid* argument was not a valid trace stream identifier.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_eventtypelist_getnext_id*() and *posix_trace_eventtypelist_rewind*() functions may be removed in a future version.

SEE ALSO

posix_trace_event( ), *posix_trace_eventid_equal*( ), *posix_trace_getnext_event*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_flush — trace stream flush from a process (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_flush(trace_id_t trid);
```

DESCRIPTION

Refer to *posix_trace_create*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_get_attr, posix_trace_get_status — retrieve the trace attributes or trace status (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_get_attr(trace_id_t trid, trace_attr_t *attr);
int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

DESCRIPTION

The *posix_trace_get_attr*() function shall copy the attributes of the active trace stream identified by *trid* into the object pointed to by the *attr* argument. If the Trace Log option is supported, *trid* may represent a pre-recorded trace log.

The *posix_trace_get_status*() function shall return, in the structure pointed to by the *statusinfo* argument, the current trace status for the trace stream identified by the *trid* argument. These status values returned in the structure pointed to by *statusinfo* shall have been appropriately read to ensure that the returned values are consistent. If the Trace Log option is supported and the *trid* argument refers to a pre-recorded trace stream, the status shall be the status of the completed trace stream.

Each time the *posix_trace_get_status*() function is used, the overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN immediately after the call completes. If the Trace Log option is supported, the *posix_trace_get_status*() function shall behave the same as when the option is not supported except for the following differences:

* If the *trid* argument refers to a trace stream with log, each time the *posix_trace_get_status*() function is used, the log overrun status of the trace stream shall be reset to POSIX_TRACE_NO_OVERRUN and the *flush_error* status shall be reset to zero immediately after the call completes.
* If the *trid* argument refers to a pre-recorded trace stream, the status returned shall be the status of the completed trace stream and the status values of the trace stream shall not be reset.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_get_attr*() function stores the trace attributes in the object pointed to by *attr*, if successful.

The *posix_trace_get_status*() function stores the trace status in the object pointed to by *statusinfo*, if successful.

ERRORS

These functions shall fail if:

EINVAL
The trace stream argument *trid* does not correspond to a valid active trace stream or a valid trace log.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_get_attr*() and *posix_trace_get_status*() functions may be removed in a future version.

SEE ALSO

posix_trace_attr_destroy( ), *posix_trace_close*( ), *posix_trace_create*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_get_filter, posix_trace_set_filter — retrieve and set the filter of an initialized trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_get_filter(trace_id_t trid, trace_event_set_t *set);
int posix_trace_set_filter(trace_id_t trid,
    const trace_event_set_t *set, int how);
```

DESCRIPTION

The *posix_trace_get_filter*() function shall retrieve, into the argument pointed to by *set*, the actual trace event filter from the trace stream specified by *trid*.

The *posix_trace_set_filter*() function shall change the set of filtered trace event types after a trace stream identified by the *trid* argument is created. This function may be called prior to starting the trace stream, or while the trace stream is active. By default, if no call is made to *posix_trace_set_filter*(), all trace events shall be recorded (that is, none of the trace event types are filtered out).

If this function is called while the trace is in progress, a special system trace event, POSIX_TRACE_FILTER, shall be recorded in the trace indicating both the old and the new sets of filtered trace event types (see *Table 2-4*, *Trace and Trace Event Filter Options: System Trace Events* and *Table 2-6*, *Trace*, *Trace Log*, *and Trace Event Filter Options: System Trace Events*).

If the *posix_trace_set_filter*() function is interrupted by a signal, an error shall be returned and the filter shall not be changed. In this case, the state of the trace stream shall not be changed.

The value of the argument *how* indicates the manner in which the set is to be changed and shall have one of the following values, as defined in the *<trace.h>* header:

POSIX_TRACE_SET_EVENTSET
: The resulting set of trace event types to be filtered shall be the trace event type set pointed to by the argument *set*.

POSIX_TRACE_ADD_EVENTSET
: The resulting set of trace event types to be filtered shall be the union of the current set and the trace event type set pointed to by the argument *set*.

POSIX_TRACE_SUB_EVENTSET
: The resulting set of trace event types to be filtered shall be all trace event types in the current set that are not in the set pointed to by the argument *set*; that is, remove each element of the specified set from the current filter.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

The *posix_trace_get_filter*() function stores the set of filtered trace event types in *set*, if successful.

ERRORS

These functions shall fail if:

EINVAL
: The value of the *trid* argument does not correspond to an active trace stream or the value of the argument pointed to by *set* is invalid.

EINTR
The operation was interrupted by a signal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_get_filter*() and *posix_trace_set_filter*() functions may be removed in a future version.

SEE ALSO

Table 2-4, *Trace and Trace Event Filter Options: System Trace Events*, *Table 2-6*, *Trace*, *Trace Log*, *and Trace Event Filter Options: System Trace Events*, *posix_trace_eventset_add*( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_get_status — retrieve the trace status (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_get_status(trace_id_t trid,
    struct posix_trace_status_info *statusinfo);
```

DESCRIPTION

Refer to *posix_trace_get_attr*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_getnext_event, posix_trace_timedgetnext_event, posix_trace_trygetnext_event — retrieve a trace event (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_getnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abstime);
int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```

DESCRIPTION

The *posix_trace_getnext_event*() function shall report a recorded trace event either from an active trace stream without log or a pre-recorded trace stream identified by the *trid* argument. The *posix_trace_trygetnext_event*() function shall report a recorded trace event from an active trace stream without log identified by the *trid* argument.

The trace event information associated with the recorded trace event shall be copied by the function into the structure pointed to by the argument *event* and the data associated with the trace event shall be copied into the buffer pointed to by the *data* argument.

The *posix_trace_getnext_event*() function shall block if the *trid* argument identifies an active trace stream and there is currently no trace event ready to be retrieved. When returning, if a recorded trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The *posix_trace_timedgetnext_event*() function shall attempt to get another trace event from an active trace stream without log, as in the *posix_trace_getnext_event*() function. However, if no trace event is available from the trace stream, the implied wait shall be terminated when the timeout specified by the argument *abstime* expires, and the function shall return the error **[ETIMEDOUT]**.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock upon which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined in the *<time.h>* header.

Under no circumstance shall the function fail with a timeout if a trace event is immediately available from the trace stream. The validity of the *abstime* argument need not be checked if a trace event is immediately available from the trace stream.

The behavior of this function for a pre-recorded trace stream is unspecified.

The *posix_trace_trygetnext_event*() function shall not block. This function shall return an error if the *trid* argument identifies a pre-recorded trace stream. If a recorded trace event was reported, the variable pointed to by the *unavailable* argument shall be set to zero. Otherwise, if no trace event was reported, the variable pointed to by the *unavailable* argument shall be set to a value different from zero.

The argument *num_bytes* shall be the size of the buffer pointed to by the *data* argument. The argument *data_len* reports to the application the length in bytes of the data record just transferred. If *num_bytes* is greater than or equal to the size of the data associated with the trace event pointed to by the *event* argument, all the recorded data shall be transferred. In this case, the *truncation-status* member of the trace event structure shall be either POSIX_TRACE_NOT_TRUNCATED, if the trace event data was recorded without truncation while tracing, or POSIX_TRACE_TRUNCATED_RECORD, if the trace event data was truncated when it was recorded. If the *num_bytes* argument is less than the length of recorded trace event data, the data transferred shall be truncated to a length of *num_bytes*, the value stored in the variable pointed to by *data_len* shall be equal to *num_bytes*, and the *truncation-status* member of the *event* structure argument shall be set to POSIX_TRACE_TRUNCATED_READ (see the **posix_trace_event_info** structure defined in <*trace.h*>).

The report of a trace event shall be sequential starting from the oldest recorded trace event. Trace events shall be reported in the order in which they were generated, up to an implementation-defined time resolution that causes the ordering of trace events occurring very close to each other to be unknown. Once reported, a trace event cannot be reported again from an active trace stream. Once a trace event is reported from an active trace stream without log, the trace stream shall make the resources associated with that trace event available to record future generated trace events.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

If successful, these functions store:

* The recorded trace event in the object pointed to by *event*
* The trace event information associated with the recorded trace event in the object pointed to by *data*
* The length of this trace event information in the object pointed to by *data_len*
* The value of zero in the object pointed to by *unavailable*

ERRORS

These functions shall fail if:

EINVAL
The trace stream identifier argument *trid* is invalid.

The *posix_trace_getnext_event*() and *posix_trace_timedgetnext_event*() functions shall fail if:

EINTR
The operation was interrupted by a signal, and so the call had no effect.

The *posix_trace_trygetnext_event*() function shall fail if:

EINVAL
The trace stream identifier argument *trid* does not correspond to an active trace stream.

The *posix_trace_timedgetnext_event*() function shall fail if:

EINVAL
There is no trace event immediately available from the trace stream, and the *timeout* argument is invalid.

ETIMEDOUT
No trace event was available from the trace stream before the specified timeout *timeout* expired.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

posix_trace_close( ), *posix_trace_create*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_open, posix_trace_rewind — trace log management (**TRACING**)

SYNOPSIS

#include <trace.h>

int posix_trace_open(int *file_desc*, trace_id_t **trid*);
int posix_trace_rewind(trace_id_t *trid*);

DESCRIPTION

Refer to *posix_trace_close*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_set_filter — set filter of an initialized trace stream (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_set_filter(trace_id_t trid,
   const trace_event_set_t *set, int how);
```

DESCRIPTION

Refer to *posix_trace_get_filter*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_shutdown — trace stream shutdown from a process (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_shutdown(trace_id_t trid);
```

DESCRIPTION

Refer to *posix_trace_create*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_start, posix_trace_stop — trace start and stop (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_start(trace_id_t trid);
int posix_trace_stop (trace_id_t trid);
```

DESCRIPTION

The *posix_trace_start*() and *posix_trace_stop*() functions, respectively, shall start and stop the trace stream identified by the argument *trid*.

The effect of calling the *posix_trace_start*() function shall be recorded in the trace stream as the POSIX_TRACE_START system trace event and the status of the trace stream shall become POSIX_TRACE_RUNNING. If the trace stream is in progress when this function is called, the POSIX_TRACE_START system trace event shall not be recorded and the trace stream shall continue to run. If the trace stream is full, the POSIX_TRACE_START system trace event shall not be recorded and the status of the trace stream shall not be changed.

The effect of calling the *posix_trace_stop*() function shall be recorded in the trace stream as the POSIX_TRACE_STOP system trace event and the status of the trace stream shall become POSIX_TRACE_SUSPENDED. If the trace stream is suspended when this function is called, the POSIX_TRACE_STOP system trace event shall not be recorded and the trace stream shall remain suspended. If the trace stream is full, the POSIX_TRACE_STOP system trace event shall not be recorded and the status of the trace stream shall not be changed.

RETURN VALUE

Upon successful completion, these functions shall return a value of zero. Otherwise, they shall return the corresponding error number.

ERRORS

These functions shall fail if:

EINVAL
The value of the argument *trid* does not correspond to an active trace stream and thus no trace stream was started or stopped.

EINTR
The operation was interrupted by a signal and thus the trace stream was not necessarily started or stopped.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *posix_trace_start*() and *posix_trace_stop*() functions may be removed in a future version.

SEE ALSO

posix_trace_create( )

The Base Definitions volume of POSIX.1-2017, **<trace.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_timedgetnext_event — retrieve a trace event (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_timedgetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable,
    const struct timespec *restrict abstime);
```

DESCRIPTION

Refer to *posix_trace_getnext_event*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_trid_eventid_open — open a trace event type identifier (**TRACING**)

SYNOPSIS

```
#include <trace.h>

int posix_trace_trid_eventid_open(trace_id_t trid,
    const char *restrict event_name,
    trace_event_id_t *restrict event);
```

DESCRIPTION

Refer to *posix_trace_eventid_equal*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_trace_trygetnext_event — retrieve a trace event (**TRACING**)

SYNOPSIS

```
#include <sys/types.h>
#include <trace.h>

int posix_trace_trygetnext_event(trace_id_t trid,
    struct posix_trace_event_info *restrict event,
    void *restrict data, size_t num_bytes,
    size_t *restrict data_len, int *restrict unavailable);
```

DESCRIPTION

Refer to *posix_trace_getnext_event*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_typed_mem_get_info — query typed memory information (**ADVANCED REALTIME**)

SYNOPSIS

```
#include <sys/mman.h>

int posix_typed_mem_get_info(int fildes,
   struct posix_typed_mem_info *info);
```

DESCRIPTION

The *posix_typed_mem_get_info*() function shall return, in the *posix_tmi_length* field of the **posix_typed_mem_info** structure pointed to by *info*, the maximum length which may be successfully allocated by the typed memory object designated by *fildes*. This maximum length shall take into account the flag POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified when the typed memory object represented by *fildes* was opened. The maximum length is dynamic; therefore, the value returned is valid only while the current mapping of the corresponding typed memory pool remains unchanged.

If *fildes* represents a typed memory object opened with neither the POSIX_TYPED_MEM_ALLOCATE flag nor the POSIX_TYPED_MEM_ALLOCATE_CONTIG flag specified, the returned value of *info*->*posix_tmi_length* is unspecified.

The *posix_typed_mem_get_info*() function may return additional implementation-defined information in other fields of the **posix_typed_mem_info** structure pointed to by *info*.

If the memory object specified by *fildes* is not a typed memory object, then the behavior of this function is undefined.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_get_info*() function shall return zero; otherwise, the corresponding error status value shall be returned.

ERRORS

The *posix_typed_mem_get_info*() function shall fail if:

EBADF
The *fildes* argument is not a valid open file descriptor.

ENODEV
The *fildes* argument is not connected to a memory object supported by this function.

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

An application that needs to allocate a block of typed memory with length dependent upon the amount of memory currently available must either query the typed memory object to obtain the amount available, or repeatedly invoke *mmap*() attempting to guess an appropriate length. While the latter method is existing practice with *malloc*(), it is awkward and imprecise. The *posix_typed_mem_get_info*() function allows an application to immediately determine available memory. This is particularly important for typed memory objects that may in some cases be scarce resources. Note that when a typed memory pool is a shared resource, some form of mutual-exclusion or synchronization may be required while typed memory is being queried and allocated to prevent race conditions.

The existing *fstat*() function is not suitable for this purpose. We realize that implementations may wish to provide other attributes of typed memory objects (for example, alignment requirements, page size, and so on). The *fstat*() function returns a structure which is not extensible and, furthermore, contains substantial information that is inappropriate for typed memory objects.

FUTURE DIRECTIONS

None.

SEE ALSO

fstat( ), *mmap*( ), *posix_typed_mem_open*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

posix_typed_mem_open — open a typed memory object (**ADVANCED REALTIME**)

SYNOPSIS

#include <sys/mman.h>

int posix_typed_mem_open(const char **name*, int *oflag*, int *tflag*);

DESCRIPTION

The *posix_typed_mem_open*() function shall establish a connection between the typed memory object specified by the string pointed to by *name* and a file descriptor. It shall create an open file description that refers to the typed memory object and a file descriptor that refers to that open file description. The file descriptor shall be allocated as described in *Section 2.14*, *File Descriptor Allocation* and can be used by other functions to refer to that typed memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *posix_typed_mem_open*() with the same value of *name* shall refer to the same typed memory object. If *name* does not begin with the <slash> character, the effect is implementation-defined.

Each typed memory object supported in a system shall be identified by a name which specifies not only its associated typed memory pool, but also the path or port by which it is accessed. That is, the same typed memory pool accessed via several different ports shall have several different corresponding names. The binding between names and typed memory objects is established in an implementation-defined manner. Unlike shared memory objects, there is no way within POSIX.1-2008 for a program to create a typed memory object.

The value of *tflag* shall determine how the typed memory object behaves when subsequently mapped by calls to *mmap*(). At most, one of the following flags defined in *<sys/mman.h>* may be specified:

POSIX_TYPED_MEM_ALLOCATE
: Allocate on *mmap*().

POSIX_TYPED_MEM_ALLOCATE_CONTIG
: Allocate contiguously on *mmap*().

POSIX_TYPED_MEM_MAP_ALLOCATABLE
: Map on *mmap*(), without affecting allocatability.

If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE specified, any subsequent call to *mmap*() using the returned file descriptor shall result in allocation and mapping of typed memory from the specified typed memory pool. The allocated memory may be a contiguous previously unallocated area of the typed memory pool or several non-contiguous previously unallocated areas (mapped to a contiguous portion of the process address space). If *tflag* has the flag POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*() using the returned file descriptor shall result in allocation and mapping of a single contiguous previously unallocated area of the typed memory pool (also mapped to a contiguous portion of the process address space). If *tflag* has none of the flags POSIX_TYPED_MEM_ALLOCATE or POSIX_TYPED_MEM_ALLOCATE_CONTIG specified, any subsequent call to *mmap*() using the returned file descriptor shall map an application-chosen area from the specified typed memory pool such that this mapped area becomes unavailable for allocation until unmapped by all processes. If *tflag* has the flag POSIX_TYPED_MEM_MAP_ALLOCATABLE specified, any subsequent call to *mmap*() using the returned file descriptor shall map an application-chosen area from the specified typed memory pool without an effect on the availability of that area for allocation; that is, mapping such an object leaves each byte of the mapped area unallocated if it was unallocated prior to the mapping or allocated if it was allocated prior to the mapping. Appropriate privileges to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag are implementation-defined.

If successful, *posix_typed_mem_open*() shall return a file descriptor for the typed memory object. The open file description is new, and therefore the file descriptor shall not share it with any other processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file descriptor shall be cleared.

The behavior of *msync*(), *ftruncate*(), and all file operations other than *mmap*(), *posix_mem_offset*(), *posix_typed_mem_get_info*(), *fstat*(), *dup*(), *dup2*(), and *close*(), is unspecified when passed a file descriptor connected to a typed memory object by this function.

The file status flags of the open file description shall be set according to the value of *oflag*. Applications shall specify exactly one of the three access mode values described below and defined in the *<fcntl.h>* header, as the value of *oflag*.

O_RDONLY	Open for read access only.
O_WRONLY	Open for write access only.
O_RDWR	Open for read or write access.

RETURN VALUE

Upon successful completion, the *posix_typed_mem_open*() function shall return a non-negative integer representing the file descriptor. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *posix_typed_mem_open*() function shall fail if:

EACCES
The typed memory object exists and the permissions specified by *oflag* are denied.

EINTR
The *posix_typed_mem_open*() operation was interrupted by a signal.

EINVAL
The flags specified in *tflag* are invalid (more than one of POSIX_TYPED_MEM_ALLOCATE, POSIX_TYPED_MEM_ALLOCATE_CONTIG, or POSIX_TYPED_MEM_MAP_ALLOCATABLE is specified).

EMFILE
All file descriptors available to the process are currently open.

ENFILE
Too many file descriptors are currently open in the system.

ENOENT
The named typed memory object does not exist.

EPERM
The caller lacks appropriate privileges to specify the POSIX_TYPED_MEM_MAP_ALLOCATABLE flag in the *tflag* argument.

The *posix_typed_mem_open*() function may fail if:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *close*( ), *dup*( ), *exec*, *fcntl*( ), *fstat*( ), *ftruncate*( ), *mmap*( ), *msync*( ), *posix_mem_offset*( ), *posix_typed_mem_get_info*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pow, powf, powl — power function

SYNOPSIS

```
#include <math.h>

double pow(double x, double y);
float powf(float x, float y);
long double powl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the value of *x* raised to the power *y*, x^y. If *x* is negative, the application shall ensure that *y* is an integer value.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the value of *x* raised to the power *y*.

For finite values of $x < 0$, and finite non-integer values of *y*, a domain error shall occur and either a NaN (if representable), or an implementation-defined value shall be returned.

If the correct value would cause overflow, a range error shall occur and *pow*(), *powf*(), and *powl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

If the correct value would cause underflow, and is not representable, a range error may occur, and *pow*(), *powf*(), and *powl*() shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

For $y < 0$, if *x* is zero, a pole error may occur and *pow*(), *powf*(), and *powl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, if *x* is ±0, a pole error shall occur and *pow*(), *powf*(), and *powl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively if *y* is an odd integer, or HUGE_VAL, HUGE_VALF, and HUGE_VALL, respectively if *y* is not an odd integer.

If *x* or *y* is a NaN, a NaN shall be returned (unless specified elsewhere in this description).

For any value of *y* (including NaN), if *x* is +1, 1.0 shall be returned.

For any value of *x* (including NaN), if *y* is ±0, 1.0 shall be returned.

For any odd integer value of $y > 0$, if *x* is ±0, ±0 shall be returned.

For $y > 0$ and not an odd integer, if *x* is ±0, +0 shall be returned.

If *x* is −1, and *y* is ±Inf, 1.0 shall be returned.

For $|x| < 1$, if *y* is −Inf, +Inf shall be returned.

For $|x| > 1$, if *y* is −Inf, +0 shall be returned.

For $|x| < 1$, if *y* is +Inf, +0 shall be returned.

For $|x| > 1$, if *y* is +Inf, +Inf shall be returned.

For *y* an odd integer < 0, if *x* is −Inf, −0 shall be returned.

For *y* < 0 and not an odd integer, if *x* is −Inf, +0 shall be returned.

For *y* an odd integer > 0, if *x* is −Inf, −Inf shall be returned.

For *y* > 0 and not an odd integer, if *x* is −Inf, +Inf shall be returned.

For *y* < 0, if *x* is +Inf, +0 shall be returned.

For *y* > 0, if *x* is +Inf, +Inf shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Domain Error
: The value of *x* is negative and *y* is a finite non-integer.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error
: The value of *x* is zero and *y* is negative.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error
: The result overflows.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Pole Error
: The value of *x* is zero and *y* is negative.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error
: The result underflows.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exp( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pread — read from a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
```

DESCRIPTION

Refer to *read*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

printf — print formatted output

SYNOPSIS

```
#include <stdio.h>

int printf(const char *restrict format, ...);
```

DESCRIPTION

Refer to *fprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pselect, select — synchronous I/O multiplexing

SYNOPSIS

```
#include <sys/select.h>

int pselect(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    const struct timespec *restrict timeout,
    const sigset_t *restrict sigmask);
int select(int nfds, fd_set *restrict readfds,
    fd_set *restrict writefds, fd_set *restrict errorfds,
    struct timeval *restrict timeout);
void FD_CLR(int fd, fd_set *fdset);
int FD_ISSET(int fd, fd_set *fdset);
void FD_SET(int fd, fd_set *fdset);
void FD_ZERO(fd_set *fdset);
```

DESCRIPTION

The *pselect*() function shall examine the file descriptor sets whose addresses are passed in the *readfds*, *writefds*, and *errorfds* parameters to see whether some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively.

The *select*() function shall be equivalent to the *pselect*() function, except as follows:

* For the *select*() function, the timeout period is given in seconds and microseconds in an argument of type **struct timeval**, whereas for the *pselect*() function the timeout period is given in seconds and nanoseconds in an argument of type **struct timespec**.
* The *select*() function has no *sigmask* argument; it shall behave as *pselect*() does when *sigmask* is a null pointer.
* Upon successful completion, the *select*() function may modify the object pointed to by the *timeout* argument.

The *pselect*() and *select*() functions shall support regular files, terminal and pseudo-terminal devices, STREAMS-based files, FIFOs, pipes, and sockets. The behavior of *pselect*() and *select*() on file descriptors that refer to other types of file is unspecified.

The *nfds* argument specifies the range of descriptors to be tested. The first *nfds* descriptors shall be checked in each set; that is, the descriptors from zero through *nfds*−1 in the descriptor sets shall be examined.

If the *readfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to read, and on output indicates which file descriptors are ready to read.

If the *writefds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for being ready to write, and on output indicates which file descriptors are ready to write.

If the *errorfds* argument is not a null pointer, it points to an object of type **fd_set** that on input specifies the file descriptors to be checked for error conditions pending, and on output indicates which file descriptors have error conditions pending.

Upon successful completion, the *pselect*() or *select*() function shall modify the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively, and shall return the total number of ready descriptors in all the output sets. For each file descriptor less than *nfds*, the corresponding bit shall be set upon successful completion if it was set on input and the associated condition is true for that file descriptor.

If none of the selected descriptors are ready for the requested operation, the *pselect*() or *select*() function shall block until at least one of the requested operations becomes ready, until the *timeout* occurs, or until interrupted by a signal. The *timeout* parameter controls how long the *pselect*() or *select*() function shall take before timing out. If the *timeout* parameter is not a null pointer, it specifies a maximum interval to wait for the selection to complete. If the specified time interval expires without any requested operation becoming ready, the function shall return. If the *timeout* parameter is a null pointer, then the call to *pselect*() or *select*() shall block indefinitely until at least one descriptor meets the specified criteria. To effect a poll, the *timeout* parameter should not be a null pointer, and should point to a zero-valued **timespec** structure.

The use of a timeout does not affect any pending timers set up by *alarm*() or *setitimer*().

Implementations may place limitations on the maximum timeout interval supported. All implementations shall support a maximum timeout interval of at least 31 days. If the *timeout* argument specifies a timeout interval greater than the implementation-defined maximum value, the maximum value shall be used as the actual timeout value. Implementations may also place limitations on the granularity of timeout intervals. If the requested timeout interval requires a finer granularity than the implementation supports, the actual timeout interval shall be rounded up to the next supported value.

If *sigmask* is not a null pointer, then the *pselect*() function shall replace the signal mask of the caller by the set of signals pointed to by *sigmask* before examining the descriptors, and shall restore the signal mask of the calling thread before returning.

A descriptor shall be considered ready for reading when a call to an input function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully. (The function might return data, an end-of-file indication, or an error other than one indicating that it is blocked, and in each of these cases the descriptor shall be considered ready for reading.)

A descriptor shall be considered ready for writing when a call to an output function with O_NONBLOCK clear would not block, whether or not the function would transfer data successfully.

If a socket has a pending error, it shall be considered to have an exceptional condition pending. Otherwise, what constitutes an exceptional condition is file type-specific. For a file descriptor for use with a socket, it is protocol-specific except as noted below. For other file types it is implementation-defined. If the operation is meaningless for a particular file type, *pselect*() or *select*() shall indicate that the descriptor is ready for read or write operations, and shall indicate that the descriptor has no exceptional condition pending.

If a descriptor refers to a socket, the implied input function is the *recvmsg*() function with parameters requesting normal and ancillary data, such that the presence of either type shall cause the socket to be marked as readable. The presence of out-of-band data shall be checked if the socket option SO_OOBINLINE has been enabled, as out-of-band data is enqueued with normal data. If the socket is currently listening, then it shall be marked as readable if an incoming connection request has been received, and a call to the *accept*() function shall complete without blocking.

If a descriptor refers to a socket, the implied output function is the *sendmsg*() function supplying an amount of normal data equal to the current value of the SO_SNDLOWAT option for the socket. If a non-blocking call to the *connect*() function has been made for a socket, and the connection attempt has either succeeded or failed leaving a pending error, the socket shall be marked as writable.

A socket shall be considered to have an exceptional condition pending if a receive operation with O_NONBLOCK clear for the open file description and with the MSG_OOB flag set would return out-of-band data without blocking. (It is protocol-specific whether the MSG_OOB flag would be used to read out-of-band data.) A socket shall also be considered to have an exceptional condition pending if an out-of-band data mark is present in the receive queue. Other circumstances under which a socket may be considered to have an exceptional condition pending are protocol-specific and implementation-defined.

If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is not a null pointer, the *pselect*() or *select*() function shall block for the time specified, or until interrupted by a signal. If the *readfds*, *writefds*, and *errorfds* arguments are all null pointers and the *timeout* argument is a null pointer, the *pselect*() or *select*() function shall block until interrupted by a signal.

File descriptors associated with regular files shall always select true for ready to read, ready to write, and error conditions.

On failure, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall not be modified. If the timeout interval expires without the specified condition being true for any of the specified file descriptors, the objects pointed to by the *readfds*, *writefds*, and *errorfds* arguments shall have all bits set to 0.

File descriptor masks of type **fd_set** can be initialized and tested with *FD_CLR*(), *FD_ISSET*(), *FD_SET*(), and *FD_ZERO*(). It is unspecified whether each of these is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined.

FD_CLR(*fd*, *fdsetp*) shall remove the file descriptor *fd* from the set pointed to by *fdsetp*. If *fd* is not a member of this set, there shall be no effect on the set, nor will an error be returned.

FD_ISSET(*fd*, *fdsetp*) shall evaluate to non-zero if the file descriptor *fd* is a member of the set pointed to by *fdsetp*, and shall evaluate to zero otherwise.

FD_SET(*fd*, *fdsetp*) shall add the file descriptor *fd* to the set pointed to by *fdsetp*. If the file descriptor *fd* is already in this set, there shall be no effect on the set, nor will an error be returned.

FD_ZERO(*fdsetp*) shall initialize the descriptor set pointed to by *fdsetp* to the null set. No error is returned if the set is not empty at the time *FD_ZERO*() is invoked.

The behavior of these macros is undefined if the *fd* argument is less than 0 or greater than or equal to FD_SETSIZE, or if *fd* is not a valid file descriptor, or if any of the arguments are expressions with side-effects.

If a thread gets canceled during a *pselect*() call, the signal mask in effect when executing the registered cleanup functions is either the original signal mask or the signal mask installed as part of the *pselect*() call.

RETURN VALUE

Upon successful completion, the *pselect*() and *select*() functions shall return the total number of bits set in the bit masks. Otherwise, −1 shall be returned, and *errno* shall be set to indicate the error.

FD_CLR(), *FD_SET*(), and *FD_ZERO*() do not return a value. *FD_ISSET*() shall return a non-zero value if the bit for the file descriptor *fd* is set in the file descriptor set pointed to by *fdset*, and 0 otherwise.

ERRORS

Under the following conditions, *pselect*() and *select*() shall fail and set *errno* to:

EBADF
One or more of the file descriptor sets specified a file descriptor that is not a valid open file descriptor.

EINTR
The function was interrupted while blocked waiting for any of the selected descriptors to become ready and before the timeout interval expired.

If SA_RESTART has been set for the interrupting signal, it is implementation-defined whether the function restarts or returns with **[EINTR]**.

EINVAL
An invalid timeout interval was specified.

EINVAL
The *nfds* argument is less than 0 or greater than FD_SETSIZE.

EINVAL
One of the specified file descriptors refers to a STREAM or multiplexer that is linked (directly or indirectly) downstream from a multiplexer.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

In earlier versions of the Single UNIX Specification, the *select*() function was defined in the *<sys/time.h>* header. This is now changed to *<sys/select.h>*. The rationale for this change was as follows: the introduction of the *pselect*() function included the *<sys/select.h>* header and the *<sys/select.h>* header defines all the related definitions for the *pselect*() and *select*() functions. Backwards-compatibility to existing XSI implementations is handled by allowing *<sys/time.h>* to include *<sys/select.h>*.

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_pselect(int nfds, fd_set *readfds, fd_set *writefds,
    fd_set errorfds, const struct timespec *timeout,
    const sigset_t *sigmask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
    pthread_cleanup_push(cleanup, &oldmask);
    result = pselect(nfds, readfds, writefds, errorfds, timeout, sigmask);
    pthread_cleanup_pop(0);
    return result;
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

accept( ), *alarm*( ), *connect*( ), *fcntl*( ), *getitimer*( ), *poll*( ), *read*( ), *recvmsg*( ), *sendmsg*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_select.h>**, **<sys_time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

psiginfo, psignal — write signal information to standard error

SYNOPSIS

```
#include <signal.h>

void psiginfo(const siginfo_t *pinfo, const char *message);
void psignal(int signum, const char *message);
```

DESCRIPTION

The *psiginfo*() and *psignal*() functions shall write a language-dependent message associated with a signal number to the standard error stream as follows:

* First, if *message* is not a null pointer and is not the empty string, the string pointed to by the *message* argument shall be written, followed by a <colon> and a <space>.
* Then the signal description string associated with *signum* or with the signal indicated by *pinfo* shall be written, followed by a <newline>.

For *psiginfo*(), the application shall ensure that the argument *pinfo* references a valid **siginfo_t** structure. For *psignal*(), if *signum* is not a valid signal number, the behavior is implementation-defined.

The *psiginfo*() and *psignal*() functions shall not change the orientation of the standard error stream.

The *psiginfo*() and *psignal*() functions shall mark for update the last data modification and last file status change timestamps of the file associated with the standard error stream at some time between their successful completion and *exit*(), *abort*(), or the completion of *fflush*() or *fclose*() on *stderr*.

The *psiginfo*() and *psignal*() functions shall not change the setting of *errno* if successful.

On error, the *psiginfo*() and *psignal*() functions shall set the error indicator for the stream to which *stderr* points, and shall set *errno* to indicate the error.

Since no value is returned, an application wishing to check for error situations should set *errno* to 0, then call *psiginfo*() or *psignal*(), then check *errno*.

RETURN VALUE

These functions shall not return a value.

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

As an alternative to setting *errno* to zero before the call and checking if it is non-zero afterwards, applications can use *ferror*() to detect whether *psiginfo*() or *psignal*() encountered an error.

An application wishing to use this method to check for error situations should call *clearerr*(*stderr*) before calling *psiginfo*() or *psignal*(), then if *ferror*(*stderr*) returns non-zero, the value of *errno* indicates which error occurred.

RATIONALE

System V historically has *psignal*() and *psiginfo*() in *<siginfo.h>*. However, the *<siginfo.h>* header is not specified in the Base Definitions volume of POSIX.1-2017, and the type **siginfo_t** is defined in *<signal.h>*.

FUTURE DIRECTIONS

None.

SEE ALSO

fputc( ), *perror*( ), *strsignal*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_atfork — register fork handlers

SYNOPSIS

```
#include <pthread.h>

int pthread_atfork(void (*prepare)(void), void (*parent)(void),
    void (*child)(void));
```

DESCRIPTION

The *pthread_atfork*() function shall declare fork handlers to be called before and after *fork*(), in the context of the thread that called *fork*(). The *prepare* fork handler shall be called before *fork*() processing commences. The *parent* fork handle shall be called after *fork*() processing completes in the parent process. The *child* fork handler shall be called after *fork*() processing completes in the child process. If no handling is desired at one or more of these three points, the corresponding fork handler address(es) may be set to NULL.

If a *fork*() call in a multi-threaded process leads to a *child* fork handler calling any function that is not async-signal-safe, the behavior is undefined.

The order of calls to *pthread_atfork*() is significant. The *parent* and *child* fork handlers shall be called in the order in which they were established by calls to *pthread_atfork*(). The *prepare* fork handlers shall be called in the opposite order.

RETURN VALUE

Upon successful completion, *pthread_atfork*() shall return a value of zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_atfork*() function shall fail if:

ENOMEM
Insufficient table space exists to record the fork handler addresses.

The *pthread_atfork*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The original usage pattern envisaged for *pthread_atfork*() was for the *prepare* fork handler to lock mutexes and other locks, and for the *parent* and *child* handlers to unlock them. However, since all of the relevant unlocking functions, except *sem_post*(), are not async-signal-safe, this usage results in undefined behavior in the child process unless the only such unlocking function it calls is *sem_post*().

RATIONALE

There are at least two serious problems with the semantics of *fork*() in a multi-threaded program. One problem has to do with state (for example, memory) covered by mutexes. Consider the case where one thread has a mutex locked and the state covered by that mutex is inconsistent while another thread calls *fork*(). In the child, the mutex is in the locked state (locked by a nonexistent thread and thus can never be unlocked). Having the child simply reinitialize the mutex is unsatisfactory since this approach does not resolve the question about how to correct or otherwise deal with the inconsistent state in the child.

It is suggested that programs that use *fork*() call an *exec* function very soon afterwards in the child process, thus resetting all states. In the meantime, only a short list of async-signal-safe library routines are promised to be available.

Unfortunately, this solution does not address the needs of multi-threaded libraries. Application programs may not be aware that a multi-threaded library is in use, and they feel free to call any number of library routines between the *fork*() and *exec* calls, just as they always have. Indeed, they may be extant single-threaded programs and cannot, therefore, be expected to obey new restrictions imposed by the threads library.

On the other hand, the multi-threaded library needs a way to protect its internal state during *fork*() in case it is re-entered later in the child process. The problem arises especially in multi-threaded I/O libraries, which are almost sure to be invoked between the *fork*() and *exec* calls to effect I/O redirection. The solution may require locking mutex variables during *fork*(), or it may entail simply resetting the state in the child after the *fork*() processing completes.

The *pthread_atfork*() function was intended to provide multi-threaded libraries with a means to protect themselves from innocent application programs that call *fork*(), and to provide multi-threaded application programs with a standard mechanism for protecting themselves from *fork*() calls in a library routine or the application itself.

The expected usage was that the prepare handler would acquire all mutex locks and the other two fork handlers would release them.

For example, an application could have supplied a prepare routine that acquires the necessary mutexes the library maintains and supplied child and parent routines that release those mutexes, thus ensuring that the child would have got a consistent snapshot of the state of the library (and that no mutexes would have been left stranded). This is good in theory, but in reality not practical. Each and every mutex and lock in the process must be located and locked. Every component of a program including third-party components must participate and they must agree who is responsible for which mutex or lock. This is especially problematic for mutexes and locks in dynamically allocated memory. All mutexes and locks internal to the implementation must be locked, too. This possibly delays the thread calling *fork*() for a long time or even indefinitely since uses of these synchronization objects may not be under control of the application. A final problem to mention here is the problem of locking streams. At least the streams under control of the system (like *stdin*, *stdout*, *stderr*) must be protected by locking the stream with *flockfile*(). But the application itself could have done that, possibly in the same thread calling *fork*(). In this case, the process will deadlock.

Alternatively, some libraries might have been able to supply just a *child* routine that reinitializes the mutexes in the library and all associated states to some known value (for example, what it was when the image was originally executed). This approach is not possible, though, because implementations are allowed to fail **_init*( ) and **_destroy*( ) calls for mutexes and locks if the mutex or lock is still locked. In this case, the *child* routine is not able to reinitialize the mutexes and locks.

When *fork*() is called, only the calling thread is duplicated in the child process. Synchronization variables remain in the same state in the child as they were in the parent at the time *fork*() was called. Thus, for example, mutex locks may be held by threads that no longer exist in the child process, and any associated states may be inconsistent. The intention was that the parent process could have avoided this by explicit code that acquires and releases locks critical to the child via *pthread_atfork*(). In addition, any critical threads would have needed to be recreated and reinitialized to the proper state in the child (also via *pthread_atfork*()).

A higher-level package may acquire locks on its own data structures before invoking lower-level packages. Under this scenario, the order specified for fork handler calls allows a simple rule of initialization for avoiding package deadlock: a package initializes all packages on which it depends before it calls the *pthread_atfork*() function for itself.

As explained, there is no suitable solution for functionality which requires non-atomic operations to be protected through mutexes and locks. This is why the POSIX.1 standard since the 1996 release requires that the child process after *fork*() in a multi-threaded process only calls async-signal-safe interfaces.

FUTURE DIRECTIONS

The *pthread_atfork*() function may be formally deprecated (for example, by shading it OB) in a future version of this standard.

SEE ALSO

atexit( ), *exec*, *fork*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_destroy, pthread_attr_init — destroy and initialize the thread attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_destroy(pthread_attr_t *attr);
int pthread_attr_init(pthread_attr_t *attr);
```

DESCRIPTION

The *pthread_attr_destroy*() function shall destroy a thread attributes object. An implementation may cause *pthread_attr_destroy*() to set *attr* to an implementation-defined invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_attr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_attr_init*() function shall initialize a thread attributes object *attr* with the default value for all of the individual attributes used by a given implementation.

The resulting attributes object (possibly modified by setting individual attribute values) when used by *pthread_create*() defines the attributes of the thread created. A single attributes object can be used in multiple simultaneous calls to *pthread_create*(). Results are undefined if *pthread_attr_init*() is called specifying an already initialized *attr* attributes object.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_destroy*() does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_destroy*() and *pthread_attr_init*() shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the thread attributes object.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Attributes objects are provided for threads, mutexes, and condition variables as a mechanism to support probable future standardization in these areas without requiring that the function itself be changed.

Attributes objects provide clean isolation of the configurable aspects of threads. For example, ‘‘stack size’’ is an important attribute of a thread, but it cannot be expressed portably. When porting a threaded program, stack sizes often need to be adjusted. The use of attributes objects can help by allowing the changes to be isolated in a single place, rather than being spread across every instance of thread creation.

Attributes objects can be used to set up ‘‘classes’ of threads with similar attributes; for example, ‘‘threads with large stacks and high priority’’ or ‘‘threads with minimal stacks’’. These classes can be defined in a single place and then referenced wherever threads need to be created. Changes to ‘‘class’’ decisions become straightforward, and detailed analysis of each *pthread_create*() call is not required.

The attributes objects are defined as opaque types as an aid to extensibility. If these objects had been specified as structures, adding new attributes would force recompilation of all multi-threaded programs when the attributes objects are extended; this might not be possible if different program components were supplied by different vendors.

Additionally, opaque attributes objects present opportunities for improving performance. Argument validity can be checked once when attributes are set, rather than each time a thread is created. Implementations often need to cache kernel objects that are expensive to create. Opaque attributes objects provide an efficient mechanism to detect when cached objects become invalid due to attribute changes.

Since assignment is not necessarily defined on a given opaque type, implementation-defined default values cannot be defined in a portable way. The solution to this problem is to allow attributes objects to be initialized dynamically by attributes object initialization functions, so that default values can be supplied automatically by the implementation.

The following proposal was provided as a suggested alternative to the supplied attributes:

1. Maintain the style of passing a parameter formed by the bitwise-inclusive OR of flags to the initialization routines (*pthread_create*(), *pthread_mutex_init*(), *pthread_cond_init*()). The parameter containing the flags should be an opaque type for extensibility. If no flags are set in the parameter, then the objects are created with default characteristics. An implementation may specify implementation-defined flag values and associated behavior.
2. If further specialization of mutexes and condition variables is necessary, implementations may specify additional procedures that operate on the **pthread_mutex_t** and **pthread_cond_t** objects (instead of on attributes objects).

The difficulties with this solution are:

1. A bitmask is not opaque if bits have to be set into bitvector attributes objects using explicitly-coded bitwise-inclusive OR operations. If the set of options exceeds an **int**, application programmers need to know the location of each bit. If bits are set or read by encapsulation (that is, get and set functions), then the bitmask is merely an implementation of attributes objects as currently defined and should not be exposed to the programmer.
2. Many attributes are not Boolean or very small integral values. For example, scheduling policy may be placed in 3-bit or 4-bit, but priority requires 5-bit or more, thereby taking up at least 8 bits out of a possible 16 bits on machines with 16-bit integers. Because of this, the bitmask can only reasonably control whether particular attributes are set or not, and it cannot serve as the repository of the value itself. The value needs to be specified as a function parameter (which is non-extensible), or by setting a structure field (which is non-opaque), or by get and set functions (making the bitmask a redundant addition to the attributes objects).

Stack size is defined as an optional attribute because the very notion of a stack is inherently machine-dependent. Some implementations may not be able to change the size of the stack, for example, and others may not need to because stack pages may be discontiguous and can be allocated and released on demand.

The attribute mechanism has been designed in large measure for extensibility. Future extensions to the attribute mechanism or to any attributes object defined in this volume of POSIX.1-2017 has to be done with care so as not to affect binary-compatibility.

Attributes objects, even if allocated by means of dynamic allocation functions such as *malloc*(), may have their size fixed at compile time. This means, for example, a *pthread_create*() in an implementation with extensions to **pthread_attr_t** cannot look beyond the area that the binary application assumes is valid. This suggests that implementations should maintain a size field in the attributes object, as well as possibly version information, if extensions in different directions (possibly by different vendors) are to be accommodated.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_destroy*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_init*() refers to an already initialized thread attributes object, it is recommended that the function should fail and report an **[EBUSY]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_getstacksize( ), *pthread_attr_getdetachstate*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getdetachstate, pthread_attr_setdetachstate — get and set the detachstate attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getdetachstate(const pthread_attr_t *attr,
    int *detachstate);
int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);
```

DESCRIPTION

The *detachstate* attribute controls whether the thread is created in a detached state. If the thread is created detached, then use of the ID of the newly created thread by the *pthread_detach*() or *pthread_join*() function is an error.

The *pthread_attr_getdetachstate*() and *pthread_attr_setdetachstate*() functions, respectively, shall get and set the *detachstate* attribute in the *attr* object.

For *pthread_attr_getdetachstate*(), *detachstate* shall be set to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

For *pthread_attr_setdetachstate*(), the application shall set *detachstate* to either PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE.

A value of PTHREAD_CREATE_DETACHED shall cause all threads created with *attr* to be in the detached state, whereas using a value of PTHREAD_CREATE_JOINABLE shall cause all threads created with *attr* to be in the joinable state. The default value of the *detachstate* attribute shall be PTHREAD_CREATE_JOINABLE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getdetachstate*() or *pthread_attr_setdetachstate*() does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_getdetachstate*() and *pthread_attr_setdetachstate*() shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getdetachstate*() function stores the value of the *detachstate* attribute in *detachstate* if successful.

ERRORS

The *pthread_attr_setdetachstate*() function shall fail if:

EINVAL
The value of *detachstate* was not valid

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

Retrieving the detachstate Attribute

This example shows how to obtain the *detachstate* attribute of a thread attribute object.

```
#include <pthread.h>

pthread_attr_t thread_attr;
int            detachstate;
int            rc;
```

```
/* code initializing thread_attr */
...

rc = pthread_attr_getdetachstate (&thread_attr, &detachstate);
if (rc!=0) {
   /* handle error */
   ...
}
else {
   /* legal values for detachstate are:
    * PTHREAD_CREATE_DETACHED or PTHREAD_CREATE_JOINABLE
    */
    ...
}
```

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getdetachstate*() or *pthread_attr_setdetachstate*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getstacksize*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getguardsize, pthread_attr_setguardsize — get and set the thread guardsize attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getguardsize(const pthread_attr_t *restrict attr,
    size_t *restrict guardsize);
int pthread_attr_setguardsize(pthread_attr_t *attr,
    size_t guardsize);
```

DESCRIPTION

The *pthread_attr_getguardsize*() function shall get the *guardsize* attribute in the *attr* object. This attribute shall be returned in the *guardsize* parameter.

The *pthread_attr_setguardsize*() function shall set the *guardsize* attribute in the *attr* object. The new value of this attribute shall be obtained from the *guardsize* parameter. If *guardsize* is zero, a guard area shall not be provided for threads created with *attr*. If *guardsize* is greater than zero, a guard area of at least size *guardsize* bytes shall be provided for each thread created with *attr*.

The *guardsize* attribute controls the size of the guard area for the created thread's stack. The *guardsize* attribute provides protection against overflow of the stack pointer. If a thread's stack is created with guard protection, the implementation allocates extra memory at the overflow end of the stack as a buffer against stack overflow of the stack pointer. If an application overflows into this buffer an error shall result (possibly in a SIGSEGV signal being delivered to the thread).

A conforming implementation may round up the value contained in *guardsize* to a multiple of the configurable system variable {PAGESIZE} (see *<sys/mman.h>*). If an implementation rounds up the value of *guardsize* to a multiple of {PAGESIZE}, a call to *pthread_attr_getguardsize*() specifying *attr* shall store in the *guardsize* parameter the guard size specified by the previous *pthread_attr_setguardsize*() function call.

The default value of the *guardsize* attribute is implementation-defined.

If the *stackaddr* attribute has been set (that is, the caller is allocating and managing its own thread stacks), the *guardsize* attribute shall be ignored and no protection shall be provided by the implementation. It is the responsibility of the application to manage stack overflow along with stack allocation and management in this case.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getguardsize*() or *pthread_attr_setguardsize*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getguardsize*() and *pthread_attr_setguardsize*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

EINVAL
The parameter *guardsize* is invalid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

Retrieving the guardsize Attribute

This example shows how to obtain the *guardsize* attribute of a thread attribute object.

```
#include <pthread.h>

pthread_attr_t thread_attr;
size_t  guardsize;
int     rc;

/* code initializing thread_attr */
...

rc = pthread_attr_getguardsize (&thread_attr, &guardsize);
if (rc != 0)  {
    /* handle error */
    ...
}
else {
    if (guardsize > 0) {
    /* a guard area of at least guardsize bytes is provided */
    ...
    }
    else {
    /* no guard area provided */
    ...
    }
}
```

APPLICATION USAGE

None.

RATIONALE

The *guardsize* attribute is provided to the application for two reasons:

1. Overflow protection can potentially result in wasted system resources. An application that creates a large number of threads, and which knows its threads never overflow their stack, can save system resources by turning off guard areas.
2. When threads allocate large data structures on the stack, large guard areas may be needed to detect stack overflow.

The default size of the guard area is left implementation-defined since on systems supporting very large page sizes, the overhead might be substantial if at least one guard page is required by default.

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getguardsize*() or *pthread_attr_setguardsize*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getinheritsched, pthread_attr_setinheritsched — get and set the inheritsched attribute (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getinheritsched(const pthread_attr_t *restrict attr,
   int *restrict inheritsched);
int pthread_attr_setinheritsched(pthread_attr_t *attr,
   int inheritsched);
```

DESCRIPTION

The *pthread_attr_getinheritsched*() and *pthread_attr_setinheritsched*() functions, respectively, shall get and set the *inheritsched* attribute in the *attr* argument.

When the attributes objects are used by *pthread_create*(), the *inheritsched* attribute determines how the other scheduling attributes of the created thread shall be set.

The supported values of *inheritsched* shall be:

PTHREAD_INHERIT_SCHED
: Specifies that the thread scheduling attributes shall be inherited from the creating thread, and the scheduling attributes in this *attr* argument shall be ignored.

PTHREAD_EXPLICIT_SCHED
: Specifies that the thread scheduling attributes shall be set to the corresponding values from this attributes object.

The symbols PTHREAD_INHERIT_SCHED and PTHREAD_EXPLICIT_SCHED are defined in the *<pthread.h>* header.

The following thread scheduling attributes defined by POSIX.1-2008 are affected by the *inheritsched* attribute: scheduling policy (*schedpolicy*), scheduling parameters (*schedparam*), and scheduling contention scope (*contentionscope*).

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getinheritsched*() or *pthread_attr_setinheritsched*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getinheritsched*() and *pthread_attr_setinheritsched*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setinheritsched*() function shall fail if:

ENOTSUP
: An attempt was made to set the attribute to an unsupported value.

The *pthread_attr_setinheritsched*() function may fail if:

EINVAL
: The value of *inheritsched* is not valid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*(). Using these routines does not affect the current running thread.

See *Section 2.9.4*, *Thread Scheduling* for further details on thread scheduling attributes and their default settings.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getinheritsched*() or *pthread_attr_setinheritsched*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getschedparam, pthread_attr_setschedparam — get and set the schedparam attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getschedparam(const pthread_attr_t *restrict attr,
    struct sched_param *restrict param);
int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

DESCRIPTION

The *pthread_attr_getschedparam*() and *pthread_attr_setschedparam*() functions, respectively, shall get and set the scheduling parameter attributes in the *attr* argument. The contents of the *param* structure are defined in the *<sched.h>* header. For the SCHED_FIFO and SCHED_RR policies, the only required member of *param* is *sched_priority*.

For the SCHED_SPORADIC policy, the required members of the *param* structure are *sched_priority*, *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, and *sched_ss_max_repl*. The specified *sched_ss_repl_period* must be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getschedparam*() or *pthread_attr_setschedparam*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getschedparam*() and *pthread_attr_setschedparam*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setschedparam*() function shall fail if:

ENOTSUP
An attempt was made to set the attribute to an unsupported value.

The *pthread_attr_setschedparam*() function may fail if:

EINVAL
The value of *param* is not valid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*(). Using these routines does not affect the current running thread.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getschedparam*() or *pthread_attr_setschedparam*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getschedpolicy, pthread_attr_setschedpolicy — get and set the schedpolicy attribute (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getschedpolicy(const pthread_attr_t *restrict attr,
   int *restrict policy);
int pthread_attr_setschedpolicy(pthread_attr_t *attr, int policy);
```

DESCRIPTION

The *pthread_attr_getschedpolicy*() and *pthread_attr_setschedpolicy*() functions, respectively, shall get and set the *schedpolicy* attribute in the *attr* argument.

The supported values of *policy* shall include SCHED_FIFO, SCHED_RR, and SCHED_OTHER, which are defined in the *<sched.h>* header. When threads executing with the scheduling policy SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on a mutex, they shall acquire the mutex in priority order when the mutex is unlocked.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getschedpolicy*() or *pthread_attr_setschedpolicy*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getschedpolicy*() and *pthread_attr_setschedpolicy*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setschedpolicy*() function shall fail if:

ENOTSUP

An attempt was made to set the attribute to an unsupported value.

The *pthread_attr_setschedpolicy*() function may fail if:

EINVAL

The value of *policy* is not valid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*(). Using these routines does not affect the current running thread.

See *Section 2.9.4*, *Thread Scheduling* for further details on thread scheduling attributes and their default settings.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getschedpolicy*() or *pthread_attr_setschedpolicy*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getscope*( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getsched-param*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getscope, pthread_attr_setscope — get and set the contentionscope attribute (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getscope(const pthread_attr_t *restrict attr,
   int *restrict contentionscope);
int pthread_attr_setscope(pthread_attr_t *attr, int contentionscope);
```

DESCRIPTION

The *pthread_attr_getscope*() and *pthread_attr_setscope*() functions, respectively, shall get and set the *contentionscope* attribute in the *attr* object.

The *contentionscope* attribute may have the values PTHREAD_SCOPE_SYSTEM, signifying system scheduling contention scope, or PTHREAD_SCOPE_PROCESS, signifying process scheduling contention scope. The symbols PTHREAD_SCOPE_SYSTEM and PTHREAD_SCOPE_PROCESS are defined in the *<pthread.h>* header.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getscope*() or *pthread_attr_setscope*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_attr_getscope*() and *pthread_attr_setscope*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_attr_setscope*() function shall fail if:

ENOTSUP
An attempt was made to set the attribute to an unsupported value.

The *pthread_attr_setscope*() function may fail if:

EINVAL
The value of *contentionscope* is not valid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

After these attributes have been set, a thread can be created with the specified attributes using *pthread_create*(). Using these routines does not affect the current running thread.

See *Section 2.9.4*, *Thread Scheduling* for further details on thread scheduling attributes and their default settings.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getscope*() or *pthread_attr_setscope*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getinheritsched*( ), *pthread_attr_getschedpolicy*( ), *pthread_attr_getschedparam*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getstack, pthread_attr_setstack — get and set stack attributes

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getstack(const pthread_attr_t *restrict attr,
    void **restrict stackaddr, size_t *restrict stacksize);
int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

DESCRIPTION

The *pthread_attr_getstack*() and *pthread_attr_setstack*() functions, respectively, shall get and set the thread creation stack attributes *stackaddr* and *stacksize* in the *attr* object.

The stack attributes specify the area of storage to be used for the created thread's stack. The base (lowest addressable byte) of the storage shall be *stackaddr*, and the size of the storage shall be *stacksize* bytes. The *stacksize* shall be at least {PTHREAD_STACK_MIN}. The *pthread_attr_setstack*() function may fail with **[EINVAL]** if *stackaddr* does not meet implementation-defined alignment requirements. All pages within the stack described by *stackaddr* and *stacksize* shall be both readable and writable by the thread.

If the *pthread_attr_getstack*() function is called before the *stackaddr* attribute has been set, the behavior is unspecified.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstack*() or *pthread_attr_setstack*() does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, these functions shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstack*() function shall store the stack attribute values in *stackaddr* and *stacksize* if successful.

ERRORS

The *pthread_attr_setstack*() function shall fail if:

EINVAL
The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds an implementation-defined limit.

The *pthread_attr_setstack*() function may fail if:

EINVAL
The value of *stackaddr* does not have proper alignment to be used as a stack, or ((**char** *)*stackaddr* + *stacksize*) lacks proper alignment.

EACCES
The stack page(s) described by *stackaddr* and *stacksize* are not both readable and writable by the thread.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

These functions are appropriate for use by applications in an environment where the stack for a thread must be placed in some particular region of memory.

While it might seem that an application could detect stack overflow by providing a protected page outside the specified stack region, this cannot be done portably. Implementations are free to place the thread's initial stack pointer anywhere within the specified region to accommodate the machine's stack pointer behavior and allocation requirements. Furthermore, on some architectures, such as the IA-64, "overflow" might mean that two separate stack pointers allocated within the region will overlap somewhere in the middle of the region.

After a successful call to *pthread_attr_setstack*(), the storage area specified by the *stackaddr* parameter is under the control of the implementation, as described in *Section 2.9.8*, *Use of Application-Managed Thread Stacks*.

The specification of the *stackaddr* attribute presents several ambiguities that make portable use of these functions impossible. For example, the standard allows implementations to impose arbitrary alignment requirements on *stackaddr*. Applications cannot assume that a buffer obtained from *malloc*() is suitably aligned. Note that although the *stacksize* value passed to *pthread_attr_setstack*() must satisfy alignment requirements, the same is not true for *pthread_attr_setstacksize*() where the implementation must increase the specified size if necessary to achieve the proper alignment.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getstack*() or *pthread_attr_setstack*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getdetachstate*( ), *pthread_attr_getstacksize*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_getstacksize, pthread_attr_setstacksize — get and set the stacksize attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_getstacksize(const pthread_attr_t *restrict attr,
    size_t *restrict stacksize);
int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

The *pthread_attr_getstacksize*() and *pthread_attr_setstacksize*() functions, respectively, shall get and set the thread creation *stacksize* attribute in the *attr* object.

The *stacksize* attribute shall define the minimum stack size (in bytes) allocated for the created threads stack.

The behavior is undefined if the value specified by the *attr* argument to *pthread_attr_getstacksize*() or *pthread_attr_setstacksize*() does not refer to an initialized thread attributes object.

RETURN VALUE

Upon successful completion, *pthread_attr_getstacksize*() and *pthread_attr_setstacksize*() shall return a value of 0; otherwise, an error number shall be returned to indicate the error.

The *pthread_attr_getstacksize*() function stores the *stacksize* attribute value in *stacksize* if successful.

ERRORS

The *pthread_attr_setstacksize*() function shall fail if:

EINVAL
The value of *stacksize* is less than {PTHREAD_STACK_MIN} or exceeds a system-imposed limit.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_attr_getstacksize*() or *pthread_attr_setstacksize*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_attr_getdetachstate*( ), *pthread_create*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_init — initialize the thread attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_init(pthread_attr_t *attr);
```

DESCRIPTION

Refer to *pthread_attr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setdetachstate — set the detachstate attribute

SYNOPSIS

#include <pthread.h>

int pthread_attr_setdetachstate(pthread_attr_t **attr*, int *detachstate*);

DESCRIPTION

Refer to *pthread_attr_getdetachstate*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setguardsize — set the thread guardsize attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_setguardsize(pthread_attr_t *attr,
   size_t guardsize);
```

DESCRIPTION

Refer to *pthread_attr_getguardsize*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setinheritsched — set the inheritsched attribute (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_setinheritsched(pthread_attr_t *attr,
   int inheritsched);
```

DESCRIPTION

Refer to *pthread_attr_getinheritsched*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setschedparam — set the schedparam attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_setschedparam(pthread_attr_t *restrict attr,
    const struct sched_param *restrict param);
```

DESCRIPTION

Refer to *pthread_attr_getschedparam*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setschedpolicy — set the schedpolicy attribute (**REALTIME THREADS**)

SYNOPSIS

#include <pthread.h>

int pthread_attr_setschedpolicy(pthread_attr_t **attr*, int *policy*);

DESCRIPTION

Refer to *pthread_attr_getschedpolicy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setscope — set the contentionscope attribute (**REALTIME THREADS**)

SYNOPSIS

#include <pthread.h>

int pthread_attr_setscope(pthread_attr_t **attr*, int *contentionscope*);

DESCRIPTION

Refer to *pthread_attr_getscope*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setstack — set the stack attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_setstack(pthread_attr_t *attr, void *stackaddr,
    size_t stacksize);
```

DESCRIPTION

Refer to *pthread_attr_getstack*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_attr_setstacksize — set the stacksize attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_attr_setstacksize(pthread_attr_t *attr, size_t stacksize);
```

DESCRIPTION

Refer to *pthread_attr_getstacksize*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrier_destroy, pthread_barrier_init — destroy and initialize a barrier object

SYNOPSIS

```
#include <pthread.h>

int pthread_barrier_destroy(pthread_barrier_t *barrier);
int pthread_barrier_init(pthread_barrier_t *restrict barrier,
   const pthread_barrierattr_t *restrict attr, unsigned count);
```

DESCRIPTION

The *pthread_barrier_destroy*() function shall destroy the barrier referenced by *barrier* and release any resources used by the barrier. The effect of subsequent use of the barrier is undefined until the barrier is reinitialized by another call to *pthread_barrier_init*(). An implementation may use this function to set *barrier* to an invalid value. The results are undefined if *pthread_barrier_destroy*() is called when any thread is blocked on the barrier, or if this function is called with an uninitialized barrier.

The *pthread_barrier_init*() function shall allocate any resources required to use the barrier referenced by *barrier* and shall initialize the barrier with attributes referenced by *attr*. If *attr* is NULL, the default barrier attributes shall be used; the effect is the same as passing the address of a default barrier attributes object. The results are undefined if *pthread_barrier_init*() is called when any thread is blocked on the barrier (that is, has not returned from the *pthread_barrier_wait*() call). The results are undefined if a barrier is used without first being initialized. The results are undefined if *pthread_barrier_init*() is called specifying an already initialized barrier.

The *count* argument specifies the number of threads that must call *pthread_barrier_wait*() before any of them successfully return from the call. The value specified by *count* must be greater than zero.

If the *pthread_barrier_init*() function fails, the barrier shall not be initialized and the contents of *barrier* are undefined.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrier_init*() function shall fail if:

EAGAIN
The system lacks the necessary resources to initialize another barrier.

EINVAL
The value specified by *count* is equal to zero.

ENOMEM
Insufficient memory exists to initialize the barrier.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_destroy*() does not refer to an initialized barrier object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *attr* argument to *pthread_barrier_init*() does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_destroy*() or *pthread_barrier_init*() refers to a barrier that is in use (for example, in a *pthread_barrier_wait*() call) by another thread, or detects that the value specified by the *barrier* argument to *pthread_barrier_init*() refers to an already initialized barrier object, it is recommended that the function should fail and report an **[EBUSY]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_wait( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrier_wait — synchronize at a barrier

SYNOPSIS

```
#include <pthread.h>

int pthread_barrier_wait(pthread_barrier_t *barrier);
```

DESCRIPTION

The *pthread_barrier_wait*() function shall synchronize participating threads at the barrier referenced by *barrier*. The calling thread shall block until the required number of threads have called *pthread_barrier_wait*() specifying the barrier.

When the required number of threads have called *pthread_barrier_wait*() specifying the barrier, the constant PTHREAD_BARRIER_SERIAL_THREAD shall be returned to one unspecified thread and zero shall be returned to each of the remaining threads. At this point, the barrier shall be reset to the state it had as a result of the most recent *pthread_barrier_init*() function that referenced it.

The constant PTHREAD_BARRIER_SERIAL_THREAD is defined in *<pthread.h>* and its value shall be distinct from any other value returned by *pthread_barrier_wait*().

The results are undefined if this function is called with an uninitialized barrier.

If a signal is delivered to a thread blocked on a barrier, upon return from the signal handler the thread shall resume waiting at the barrier if the barrier wait has not completed (that is, if the required number of threads have not arrived at the barrier during the execution of the signal handler); otherwise, the thread shall continue as normal from the completed barrier wait. Until the thread in the signal handler returns from it, it is unspecified whether other threads may proceed past the barrier once they have all reached it.

A thread that has blocked on a barrier shall not prevent any unblocked thread that is eligible to use the same processing resources from eventually making forward progress in its execution. Eligibility for processing resources shall be determined by the scheduling policy.

RETURN VALUE

Upon successful completion, the *pthread_barrier_wait*() function shall return PTHREAD_BARRIER_SERIAL_THREAD for a single (arbitrary) thread synchronized at the barrier and zero for each of the other threads. Otherwise, an error number shall be returned to indicate the error.

ERRORS

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *barrier* argument to *pthread_barrier_wait*() does not refer to an initialized barrier object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_destroy( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrierattr_destroy, pthread_barrierattr_init — destroy and initialize the barrier attributes object

SYNOPSIS

#include <pthread.h>

int pthread_barrierattr_destroy(pthread_barrierattr_t **attr*);
int pthread_barrierattr_init(pthread_barrierattr_t **attr*);

DESCRIPTION

The *pthread_barrierattr_destroy*() function shall destroy a barrier attributes object. A destroyed *attr* attributes object can be reinitialized using *pthread_barrierattr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause *pthread_barrierattr_destroy*() to set the object referenced by *attr* to an invalid value.

The *pthread_barrierattr_init*() function shall initialize a barrier attributes object *attr* with the default value for all of the attributes defined by the implementation.

If *pthread_barrierattr_init*() is called specifying an already initialized *attr* attributes object, the results are undefined.

After a barrier attributes object has been used to initialize one or more barriers, any function affecting the attributes object (including destruction) shall not affect any previously initialized barrier.

The behavior is undefined if the value specified by the *attr* argument to *pthread_barrierattr_destroy*() does not refer to an initialized barrier attributes object.

RETURN VALUE

If successful, the *pthread_barrierattr_destroy*() and *pthread_barrierattr_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrierattr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the barrier attributes object.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_barrierattr_destroy*() does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrierattr_getpshared( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrierattr_getpshared, pthread_barrierattr_setpshared — get and set the process-shared attribute of the barrier attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_barrierattr_getpshared(const pthread_barrierattr_t
    *restrict attr, int *restrict pshared);
int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_barrierattr_getpshared*() function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*. The *pthread_barrierattr_setpshared*() function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a barrier to be operated upon by any thread that has access to the memory where the barrier is allocated. See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE. Both constants PTHREAD_PROCESS_SHARED and PTHREAD_PROCESS_PRIVATE are defined in *<pthread.h>*.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_barrierattr_getpshared*() or *pthread_barrierattr_setpshared*() does not refer to an initialized barrier attributes object.

RETURN VALUE

If successful, the *pthread_barrierattr_getpshared*() function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_barrierattr_setpshared*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_barrierattr_setpshared*() function may fail if:

EINVAL
The new value specified for the *process-shared* attribute is not one of the legal values PTHREAD_PROCESS_SHARED or PTHREAD_PROCESS_PRIVATE.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_barrierattr_getpshared*() and *pthread_barrierattr_setpshared*() functions are part of the Thread Process-Shared Synchronization option and need not be provided on all implementations.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_barrierattr_getpshared*() or *pthread_barrierattr_setpshared*() does not refer to an initialized barrier attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_barrier_destroy( ), *pthread_barrierattr_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrierattr_init — initialize the barrier attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_barrierattr_init(pthread_barrierattr_t *attr);
```

DESCRIPTION

Refer to *pthread_barrierattr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_barrierattr_setpshared — set the process-shared attribute of the barrier attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_barrierattr_setpshared(pthread_barrierattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_barrierattr_getpshared*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cancel — cancel execution of a thread

SYNOPSIS

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

DESCRIPTION

The *pthread_cancel*() function shall request that *thread* be canceled. The target thread's cancelability state and type determines when the cancellation takes effect. When the cancellation is acted on, the cancellation cleanup handlers for *thread* shall be called. When the last cancellation cleanup handler returns, the thread-specific data destructor functions shall be called for *thread*. When the last destructor function returns, *thread* shall be terminated.

The cancellation processing in the target thread shall run asynchronously with respect to the calling thread returning from *pthread_cancel*().

RETURN VALUE

If successful, the *pthread_cancel*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_cancel*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Two alternative functions were considered for sending the cancellation notification to a thread. One would be to define a new SIGCANCEL signal that had the cancellation semantics when delivered; the other was to define the new *pthread_cancel*() function, which would trigger the cancellation semantics.

The advantage of a new signal was that so much of the delivery criteria were identical to that used when trying to deliver a signal that making cancellation notification a signal was seen as consistent. Indeed, many implementations implement cancellation using a special signal. On the other hand, there would be no signal functions that could be used with this signal except *pthread_kill*(), and the behavior of the delivered cancellation signal would be unlike any previously existing defined signal.

The benefits of a special function include the recognition that this signal would be defined because of the similar delivery criteria and that this is the only common behavior between a cancellation request and a signal. In addition, the cancellation delivery mechanism does not have to be implemented as a signal. There are also strong, if not stronger, parallels with language exception mechanisms than with signals that are potentially obscured if the delivery mechanism is visibly closer to signals.

In the end, it was considered that as there were so many exceptions to the use of the new signal with existing signals functions it would be misleading. A special function has resolved this problem. This function was carefully defined so that an implementation wishing to provide the cancellation functions on top of signals could do so. The special function also means that implementations are not obliged to implement cancellation with signals.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_exit( ), *pthread_cond_timedwait*( ), *pthread_join*( ), *pthread_setcancelstate*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cleanup_pop, pthread_cleanup_push — establish cancellation handlers

SYNOPSIS

#include <pthread.h>

void pthread_cleanup_pop(int *execute*);
void pthread_cleanup_push(void (**routine*)(void*), void **arg*);

DESCRIPTION

The *pthread_cleanup_pop*() function shall remove the routine at the top of the calling thread's cancellation cleanup stack and optionally invoke it (if *execute* is non-zero).

The *pthread_cleanup_push*() function shall push the specified cancellation cleanup handler *routine* onto the calling thread's cancellation cleanup stack. The cancellation cleanup handler shall be popped from the cancellation cleanup stack and invoked with the argument *arg* when:

* The thread exits (that is, calls *pthread_exit*()).
* The thread acts upon a cancellation request.
* The thread calls *pthread_cleanup_pop*() with a non-zero *execute* argument.

It is unspecified whether *pthread_cleanup_push*() and *pthread_cleanup_pop*() are macros or functions. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with any of these names, the behavior is undefined. The application shall ensure that they appear as statements, and in pairs within the same lexical scope (that is, the *pthread_cleanup_push*() macro may be thought to expand to a token list whose first token is **'{'** with *pthread_cleanup_pop*() expanding to a token list whose last token is the corresponding **'}'**).

The effect of calling *longjmp*() or *siglongjmp*() is undefined if there have been any calls to *pthread_cleanup_push*() or *pthread_cleanup_pop*() made without the matching call since the jump buffer was filled. The effect of calling *longjmp*() or *siglongjmp*() from inside a cancellation cleanup handler is also undefined unless the jump buffer was also filled in the cancellation cleanup handler.

The effect of the use of **return**, **break**, **continue**, and **goto** to prematurely leave a code block described by a pair of *pthread_cleanup_push*() and *pthread_cleanup_pop*() functions calls is undefined.

RETURN VALUE

The *pthread_cleanup_push*() and *pthread_cleanup_pop*() functions shall not return a value.

ERRORS

No errors are defined.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

The following is an example using thread primitives to implement a cancelable, writers-priority read-write lock:

```
typedef struct {
    pthread_mutex_t lock;
    pthread_cond_t rcond,
        wcond;
    int lock_count; /* < 0 .. Held by writer. */
                    /* > 0 .. Held by lock_count readers. */
```

```
                /* = 0 .. Held by nobody. */
    int waiting_writers; /* Count of waiting writers. */
} rwlock;

void
waiting_reader_cleanup(void *arg)
{
    rwlock *l;

    l = (rwlock *) arg;
    pthread_mutex_unlock(&l->lock);
}

void
lock_for_read(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    pthread_cleanup_push(waiting_reader_cleanup, l);
    while ((l->lock_count < 0) || (l->waiting_writers != 0))
        pthread_cond_wait(&l->rcond, &l->lock);
    l->lock_count++;
   /*
    * Note the pthread_cleanup_pop executes
    * waiting_reader_cleanup.
    */
    pthread_cleanup_pop(1);
}

void
release_read_lock(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    if (--l->lock_count == 0)
        pthread_cond_signal(&l->wcond);
    pthread_mutex_unlock(&l->lock);
}

void
waiting_writer_cleanup(void *arg)
{
    rwlock *l;

    l = (rwlock *) arg;
    if ((--l->waiting_writers == 0) && (l->lock_count >= 0)) {
       /*
        * This only happens if we have been canceled. If the
        * lock is not held by a writer, there may be readers who
        * were blocked because waiting_writers was positive; they
        * can now be unblocked.
        */
        pthread_cond_broadcast(&l->rcond);
    }
    pthread_mutex_unlock(&l->lock);
}

void
lock_for_write(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    l->waiting_writers++;
    pthread_cleanup_push(waiting_writer_cleanup, l);
    while (l->lock_count != 0)
        pthread_cond_wait(&l->wcond, &l->lock);
    l->lock_count = -1;
   /*
    * Note the pthread_cleanup_pop executes
    * waiting_writer_cleanup.
    */
    pthread_cleanup_pop(1);
}
void
release_write_lock(rwlock *l)
{
    pthread_mutex_lock(&l->lock);
    l->lock_count = 0;
    if (l->waiting_writers == 0)
        pthread_cond_broadcast(&l->rcond);
    else
        pthread_cond_signal(&l->wcond);
    pthread_mutex_unlock(&l->lock);
}
/*
 * This function is called to initialize the read/write lock.
 */
void
initialize_rwlock(rwlock *l)
{
    pthread_mutex_init(&l->lock, pthread_mutexattr_default);
    pthread_cond_init(&l->wcond, pthread_condattr_default);
    pthread_cond_init(&l->rcond, pthread_condattr_default);
    l->lock_count = 0;
    l->waiting_writers = 0;
}
reader_thread()
{
    lock_for_read(&lock);
    pthread_cleanup_push(release_read_lock, &lock);
   /*
    * Thread has read lock.
    */
    pthread_cleanup_pop(1);
}
writer_thread()
{
    lock_for_write(&lock);
    pthread_cleanup_push(release_write_lock, &lock);
   /*
    * Thread has write lock.
    */
pthread_cleanup_pop(1);
}
```

APPLICATION USAGE

The two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*() and *pthread_cleanup_pop*(), can be thought of as left and right-parentheses. They always need to be matched.

RATIONALE

The restriction that the two routines that push and pop cancellation cleanup handlers, *pthread_cleanup_push*() and *pthread_cleanup_pop*(), have to appear in the same lexical scope allows for efficient macro or compiler implementations and efficient storage management. A sample implementation of these routines as macros might look like this:

```
#define pthread_cleanup_push(rtn,arg) { \
    struct _pthread_handler_rec __cleanup_handler, **__head; \
    __cleanup_handler.rtn = rtn; \
    __cleanup_handler.arg = arg; \
    (void) pthread_getspecific(_pthread_handler_key, &__head); \
    __cleanup_handler.next = *__head; \
    *__head = &__cleanup_handler;

#define pthread_cleanup_pop(ex) \
    *__head = __cleanup_handler.next; \
    if (ex) (*__cleanup_handler.rtn)(__cleanup_handler.arg); \
}
```

A more ambitious implementation of these routines might do even better by allowing the compiler to note that the cancellation cleanup handler is a constant and can be expanded inline.

This volume of POSIX.1-2017 currently leaves unspecified the effect of calling *longjmp*() from a signal handler executing in a POSIX System Interfaces function. If an implementation wants to allow this and give the programmer reasonable behavior, the *longjmp*() function has to call all cancellation cleanup handlers that have been pushed but not popped since the time *setjmp*() was called.

Consider a multi-threaded function called by a thread that uses signals. If a signal were delivered to a signal handler during the operation of *qsort*() and that handler were to call *longjmp*() (which, in turn, did *not* call the cancellation cleanup handlers) the helper threads created by the *qsort*() function would not be canceled. Instead, they would continue to execute and write into the argument array even though the array might have been popped off the stack.

Note that the specified cleanup handling mechanism is especially tied to the C language and, while the requirement for a uniform mechanism for expressing cleanup is language-independent, the mechanism used in other languages may be quite different. In addition, this mechanism is really only necessary due to the lack of a real exception mechanism in the C language, which would be the ideal solution.

There is no notion of a cancellation cleanup-safe function. If an application has no cancellation points in its signal handlers, blocks any signal whose handler may have cancellation points while calling async-unsafe functions, or disables cancellation while calling async-unsafe functions, all functions may be safely called from cancellation cleanup routines.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cancel( ), *pthread_setcancelstate*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cond_broadcast, pthread_cond_signal — broadcast or signal a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_broadcast(pthread_cond_t *cond);
int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

These functions shall unblock threads blocked on a condition variable.

The *pthread_cond_broadcast*() function shall unblock all threads currently blocked on the specified condition variable *cond*.

The *pthread_cond_signal*() function shall unblock at least one of the threads that are blocked on the specified condition variable *cond* (if any threads are blocked on *cond*).

If more than one thread is blocked on a condition variable, the scheduling policy shall determine the order in which threads are unblocked. When each thread unblocked as a result of a *pthread_cond_broadcast*() or *pthread_cond_signal*() returns from its call to *pthread_cond_wait*() or *pthread_cond_timedwait*(), the thread shall own the mutex with which it called *pthread_cond_wait*() or *pthread_cond_timedwait*(). The thread(s) that are unblocked shall contend for the mutex according to the scheduling policy (if applicable), and as if each had called *pthread_mutex_lock*().

The *pthread_cond_broadcast*() or *pthread_cond_signal*() functions may be called by a thread whether or not it currently owns the mutex that threads calling *pthread_cond_wait*() or *pthread_cond_timedwait*() have associated with the condition variable during their waits; however, if predictable scheduling behavior is required, then that mutex shall be locked by the thread calling *pthread_cond_broadcast*() or *pthread_cond_signal*().

The *pthread_cond_broadcast*() and *pthread_cond_signal*() functions shall have no effect if there are no threads currently blocked on *cond*.

The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_broadcast*() or *pthread_cond_signal*() does not refer to an initialized condition variable.

RETURN VALUE

If successful, the *pthread_cond_broadcast*() and *pthread_cond_signal*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_cond_broadcast*() function is used whenever the shared-variable state has been changed in a way that more than one thread can proceed with its task. Consider a single producer/multiple consumer problem, where the producer can insert multiple items on a list that is accessed one item at a time by the consumers. By calling the *pthread_cond_broadcast*() function, the producer would notify all consumers that might be waiting, and thereby the application would receive more throughput on a multi-processor. In addition, *pthread_cond_broadcast*() makes it easier to implement a read-write lock. The *pthread_cond_broadcast*() function is needed in order to wake up all waiting readers when a writer releases its lock. Finally, the two-phase commit algorithm can use this broadcast function to notify all clients of an impending transaction commit.

It is not safe to use the *pthread_cond_signal*() function in a signal handler that is invoked asynchronously. Even if it were safe, there would still be a race between the test of the Boolean *pthread_cond_wait*() that could not be efficiently eliminated.

Mutexes and condition variables are thus not suitable for releasing a waiting thread by signaling from code running in a signal handler.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_broadcast*() or *pthread_cond_signal*() does not refer to an initialized condition variable, it is recommended that the function should fail and report an **[EINVAL]** error.

Multiple Awakenings by Condition Signal

On a multi-processor, it may be impossible for an implementation of *pthread_cond_signal*() to avoid the unblocking of more than one thread blocked on a condition variable. For example, consider the following partial implementation of *pthread_cond_wait*() and *pthread_cond_signal*(), executed by two threads in the order given. One thread is trying to wait on the condition variable, another is concurrently executing *pthread_cond_signal*(), while a third thread is already waiting.

```
pthread_cond_wait(mutex, cond):
    value = cond->value; /* 1 */
    pthread_mutex_unlock(mutex); /* 2 */
    pthread_mutex_lock(cond->mutex); /* 10 */
    if (value == cond->value) { /* 11 */
        me->next_cond = cond->waiter;
        cond->waiter = me;
        pthread_mutex_unlock(cond->mutex);
        unable_to_run(me);
    } else
        pthread_mutex_unlock(cond->mutex); /* 12 */
    pthread_mutex_lock(mutex); /* 13 */

pthread_cond_signal(cond):
    pthread_mutex_lock(cond->mutex); /* 3 */
    cond->value++; /* 4 */
    if (cond->waiter) { /* 5 */
        sleeper = cond->waiter; /* 6 */
        cond->waiter = sleeper->next_cond; /* 7 */
        able_to_run(sleeper); /* 8 */
    }
    pthread_mutex_unlock(cond->mutex); /* 9 */
```

The effect is that more than one thread can return from its call to *pthread_cond_wait*() or *pthread_cond_timedwait*() as a result of one call to *pthread_cond_signal*(). This effect is called "spurious wakeup". Note that the situation is self-correcting in that the number of threads that are so awakened is finite; for example, the next thread to call *pthread_cond_wait*() after the sequence of events above blocks.

While this problem could be resolved, the loss of efficiency for a fringe condition that occurs only rarely is unacceptable, especially given that one has to check the predicate associated with a condition variable anyway. Correcting this problem would unnecessarily reduce the degree of concurrency in this basic building block for all higher-level synchronization operations.

An added benefit of allowing spurious wakeups is that applications are forced to code a predicate-testing-loop around the condition wait. This also makes the application tolerate superfluous condition broadcasts or signals on the same condition variable that may be coded in some other part of the application. The resulting applications are thus more robust. Therefore, POSIX.1-2008 explicitly documents that spurious wakeups may occur.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_cond_timedwait*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cond_destroy, pthread_cond_init — destroy and initialize condition variables

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_destroy(pthread_cond_t *cond);
int pthread_cond_init(pthread_cond_t *restrict cond,
   const pthread_condattr_t *restrict attr);
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
```

DESCRIPTION

The *pthread_cond_destroy*() function shall destroy the given condition variable specified by *cond*; the object becomes, in effect, uninitialized. An implementation may cause *pthread_cond_destroy*() to set the object referenced by *cond* to an invalid value. A destroyed condition variable object can be reinitialized using *pthread_cond_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized condition variable upon which no threads are currently blocked. Attempting to destroy a condition variable upon which other threads are currently blocked results in undefined behavior.

The *pthread_cond_init*() function shall initialize the condition variable referenced by *cond* with attributes referenced by *attr*. If *attr* is NULL, the default condition variable attributes shall be used; the effect is the same as passing the address of a default condition variable attributes object. Upon successful initialization, the state of the condition variable shall become initialized.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

Attempting to initialize an already initialized condition variable results in undefined behavior.

In cases where default condition variable attributes are appropriate, the macro PTHREAD_COND_INITIALIZER can be used to initialize condition variables. The effect shall be equivalent to dynamic initialization by a call to *pthread_cond_init*() with parameter *attr* specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *cond* argument to *pthread_cond_destroy*() does not refer to an initialized condition variable.

The behavior is undefined if the value specified by the *attr* argument to *pthread_cond_init*() does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_cond_destroy*() and *pthread_cond_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_cond_init*() function shall fail if:

EAGAIN
The system lacked the necessary resources (other than memory) to initialize another condition variable.

ENOMEM
Insufficient memory exists to initialize the condition variable.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

A condition variable can be destroyed immediately after all the threads that are blocked on it are awakened. For example, consider the following code:

```
struct list {
    pthread_mutex_t lm;
    ...
}

struct elt {
    key k;
    int busy;
    pthread_cond_t notbusy;
    ...
}

/* Find a list element and reserve it. */
struct elt *
list_find(struct list *lp, key k)
{
    struct elt *ep;

    pthread_mutex_lock(&lp->lm);
    while ((ep = find_elt(l, k) != NULL) && ep->busy)
        pthread_cond_wait(&ep->notbusy, &lp->lm);
    if (ep != NULL)
        ep->busy = 1;
    pthread_mutex_unlock(&lp->lm);
    return(ep);
}

delete_elt(struct list *lp, struct elt *ep)
{
    pthread_mutex_lock(&lp->lm);
    assert(ep->busy);
    ... remove ep from list ...
    ep->busy = 0;  /* Paranoid. */
(A) pthread_cond_broadcast(&ep->notbusy);
    pthread_mutex_unlock(&lp->lm);
(B) pthread_cond_destroy(&ep->notbusy);
    free(ep);
}
```

In this example, the condition variable and its list element may be freed (line B) immediately after all threads waiting for it are awakened (line A), since the mutex and the code ensure that no other thread can touch the element to be deleted.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_destroy*() does not refer to an initialized condition variable, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_destroy*() or *pthread_cond_init*() refers to a condition variable that is in use (for example, in a *pthread_cond_wait*() call) by another thread, or detects that the value specified by the *cond* argument to *pthread_cond_init*() refers to an already initialized condition variable, it is recommended that the function should fail and report an **[EBUSY]** error.

If an implementation detects that the value specified by the *attr* argument to *pthread_cond_init*() does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

See also *pthread_mutex_destroy*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_broadcast( ), *pthread_cond_timedwait*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cond_signal — signal a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_signal(pthread_cond_t *cond);
```

DESCRIPTION

Refer to *pthread_cond_broadcast*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_cond_timedwait, pthread_cond_wait — wait on a condition

SYNOPSIS

```
#include <pthread.h>

int pthread_cond_timedwait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex,
    const struct timespec *restrict abstime);
int pthread_cond_wait(pthread_cond_t *restrict cond,
    pthread_mutex_t *restrict mutex);
```

DESCRIPTION

The *pthread_cond_timedwait*() and *pthread_cond_wait*() functions shall block on a condition variable. The application shall ensure that these functions are called with *mutex* locked by the calling thread; otherwise, an error (for PTHREAD_MUTEX_ERRORCHECK and robust mutexes) or undefined behavior (for other mutexes) results.

These functions atomically release *mutex* and cause the calling thread to block on the condition variable *cond*; atomically here means "atomically with respect to access by another thread to the mutex and then the condition variable". That is, if another thread is able to acquire the mutex after the about-to-block thread has released it, then a subsequent call to *pthread_cond_broadcast*() or *pthread_cond_signal*() in that thread shall behave as if it were issued after the about-to-block thread has blocked.

Upon successful return, the mutex shall have been locked and shall be owned by the calling thread. If *mutex* is a robust mutex where an owner terminated while holding the lock and the state is recoverable, the mutex shall be acquired even though the function returns an error code.

When using condition variables there is always a Boolean predicate involving shared variables associated with each condition wait that is true if the thread should proceed. Spurious wakeups from the *pthread_cond_timedwait*() or *pthread_cond_wait*() functions may occur. Since the return from *pthread_cond_timedwait*() or *pthread_cond_wait*() does not imply anything about the value of this predicate, the predicate should be re-evaluated upon such return.

When a thread waits on a condition variable, having specified a particular mutex to either the *pthread_cond_timedwait*() or the *pthread_cond_wait*() operation, a dynamic binding is formed between that mutex and condition variable that remains in effect as long as at least one thread is blocked on the condition variable. During this time, the effect of an attempt by any thread to wait on that condition variable using a different mutex is undefined. Once all waiting threads have been unblocked (as by the *pthread_cond_broadcast*() operation), the next wait operation on that condition variable shall form a new dynamic binding with the mutex specified by that wait operation. Even though the dynamic binding between condition variable and mutex may be removed or replaced between the time a thread is unblocked from a wait on the condition variable and the time that it returns to the caller or begins cancellation cleanup, the unblocked thread shall always re-acquire the mutex specified in the condition wait operation call from which it is returning.

A condition wait (whether timed or not) is a cancellation point. When the cancelability type of a thread is set to PTHREAD_CANCEL_DEFERRED, a side-effect of acting upon a cancellation request while in a condition wait is that the mutex is (in effect) re-acquired before calling the first cancellation cleanup handler. The effect is as if the thread were unblocked, allowed to execute up to the point of returning from the call to *pthread_cond_timedwait*() or *pthread_cond_wait*(), but at that point notices the cancellation request and instead of returning to the caller of *pthread_cond_timedwait*() or *pthread_cond_wait*(), starts the thread cancellation activities, which includes calling cancellation cleanup handlers.

A thread that has been unblocked because it has been canceled while blocked in a call to *pthread_cond_timedwait*() or *pthread_cond_wait*() shall not consume any condition signal that may be directed concurrently at the condition variable if there are other threads blocked on the condition variable.

The *pthread_cond_timedwait*() function shall be equivalent to *pthread_cond_wait*(), except that an error is returned if the absolute time specified by *abstime* passes (that is, system time equals or exceeds *abstime*) before the condition *cond* is signaled or broadcasted, or if the absolute time specified by *abstime* has already been passed at the time of the call. When such timeouts occur, *pthread_cond_timedwait*() shall nonetheless release and re-acquire the mutex referenced by *mutex*, and may consume a condition signal directed concurrently at the condition variable.

The condition variable shall have a clock attribute which specifies the clock that shall be used to measure the time specified by the *abstime* argument. The *pthread_cond_timedwait*() function is also a cancellation point.

If a signal is delivered to a thread waiting for a condition variable, upon return from the signal handler the thread resumes waiting for the condition variable as if it was not interrupted, or it shall return zero due to spurious wakeup.

The behavior is undefined if the value specified by the *cond* or *mutex* argument to these functions does not refer to an initialized condition variable or an initialized mutex object, respectively.

RETURN VALUE

Except for **[ETIMEDOUT]**, **[ENOTRECOVERABLE]**, and **[EOWNERDEAD]**, all these error checks shall act as if they were performed immediately at the beginning of processing for the function and shall cause an error return, in effect, prior to modifying the state of the mutex specified by *mutex* or the condition variable specified by *cond*.

Upon successful completion, a value of zero shall be returned; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

ENOTRECOVERABLE
The state protected by the mutex is not recoverable.

EOWNERDEAD
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

EPERM
The mutex type is PTHREAD_MUTEX_ERRORCHECK or the mutex is a robust mutex, and the current thread does not own the mutex.

The *pthread_cond_timedwait*() function shall fail if:

ETIMEDOUT
The time specified by *abstime* to *pthread_cond_timedwait*() has passed.

EINVAL
The *abstime* argument specified a nanosecond value less than zero or greater than or equal to 1000 million.

These functions may fail if:

EOWNERDEAD
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is **[EOWNERDEAD]**. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

If an implementation detects that the value specified by the *cond* argument to *pthread_cond_timedwait*() or *pthread_cond_wait*() does not refer to an initialized condition variable, or detects that the value specified by the *mutex* argument to *pthread_cond_timedwait*() or *pthread_cond_wait*() does not refer to an initialized mutex object, it is recommended that the function should fail and report an **[EINVAL]** error.

Condition Wait Semantics

It is important to note that when *pthread_cond_wait*() and *pthread_cond_timedwait*() return without error, the associated predicate may still be false. Similarly, when *pthread_cond_timedwait*() returns with the timeout error, the associated predicate may be true due to an unavoidable race between the expiration of the timeout and the predicate state change.

The application needs to recheck the predicate on any return because it cannot be sure there is another thread waiting on the thread to handle the signal, and if there is not then the signal is lost. The burden is on the application to check the predicate.

Some implementations, particularly on a multi-processor, may sometimes cause multiple threads to wake up when the condition variable is signaled simultaneously on different processors.

In general, whenever a condition wait returns, the thread has to re-evaluate the predicate associated with the condition wait to determine whether it can safely proceed, should wait again, or should declare a timeout. A return from the wait does not imply that the associated predicate is either true or false.

It is thus recommended that a condition wait be enclosed in the equivalent of a "while loop" that checks the predicate.

Timed Wait Semantics

An absolute time measure was chosen for specifying the timeout parameter for two reasons. First, a relative time measure can be easily implemented on top of a function that specifies absolute time, but there is a race condition associated with specifying an absolute timeout on top of a function that specifies relative timeouts. For example, assume that *clock_gettime*() returns the current time and *cond_relative_timed_wait*() uses relative timeouts:

```
clock_gettime(CLOCK_REALTIME, &now)
reltime = sleep_til_this_absolute_time -now;
cond_relative_timed_wait(c, m, &reltime);
```

If the thread is preempted between the first statement and the last statement, the thread blocks for too long. Blocking, however, is irrelevant if an absolute timeout is used. An absolute timeout also need not be recomputed if it is used multiple times in a loop, such as that enclosing a condition wait.

For cases when the system clock is advanced discontinuously by an operator, it is expected that implementations process any timed wait expiring at an intervening time as if that time had actually occurred.

Cancellation and Condition Wait

A condition wait, whether timed or not, is a cancellation point. That is, the functions *pthread_cond_wait*() or *pthread_cond_timedwait*() are points where a pending (or concurrent) cancellation request is noticed. The reason for this is that an indefinite wait is possible at these points—whatever event is being waited for, even if the program is totally correct, might never occur; for example, some input data being awaited might never be sent. By making condition wait a cancellation point, the thread can be canceled and perform its cancellation cleanup handler even though it may be stuck in some indefinite wait.

A side-effect of acting on a cancellation request while a thread is blocked on a condition variable is to re-acquire the mutex before calling any of the cancellation cleanup handlers. This is done in order to ensure that the cancellation cleanup handler is executed in the same state as the critical code that lies both before and after the call to the condition wait function. This rule is also required when interfacing to POSIX threads from languages, such as Ada or C++, which may choose to map cancellation onto a language exception; this rule ensures that each exception handler guarding a critical section can always safely depend upon the fact that the associated mutex has already been locked regardless of exactly where within the critical section the exception was raised. Without this rule, there would not be a uniform rule that exception handlers could follow regarding the lock, and so coding would become very cumbersome.

Therefore, since *some* statement has to be made regarding the state of the lock when a cancellation is delivered during a wait, a definition has been chosen that makes application coding most convenient and error free.

When acting on a cancellation request while a thread is blocked on a condition variable, the implementation is required to ensure that the thread does not consume any condition signals directed at that condition variable if there are any other threads waiting on that condition variable. This rule is specified in order to avoid deadlock conditions that could occur if these two independent requests (one acting on a thread and the other acting on the condition variable) were not processed independently.

Performance of Mutexes and Condition Variables

Mutexes are expected to be locked only for a few instructions. This practice is almost automatically enforced by the desire of programmers to avoid long serial regions of execution (which would reduce total effective parallelism).

When using mutexes and condition variables, one tries to ensure that the usual case is to lock the mutex, access shared data, and unlock the mutex. Waiting on a condition variable should be a relatively rare situation. For example, when implementing a read-write lock, code that acquires a read-lock typically needs only to increment the count of readers (under mutual-exclusion) and return. The calling thread would actually wait on the condition variable only when there is already an active writer. So the efficiency of a synchronization operation is bounded by the cost of mutex lock/unlock and not by condition wait. Note that in the usual case there is no context switch.

This is not to say that the efficiency of condition waiting is unimportant. Since there needs to be at least one context switch per Ada rendezvous, the efficiency of waiting on a condition variable is important. The cost of waiting on a condition variable should be little more than the minimal cost for a context switch plus the time to unlock and lock the mutex.

Features of Mutexes and Condition Variables

It had been suggested that the mutex acquisition and release be decoupled from condition wait. This was rejected because it is the combined nature of the operation that, in fact, facilitates realtime implementations. Those implementations can atomically move a high-priority thread between the condition variable and the mutex in a manner that is transparent to the caller. This can prevent extra context switches and provide more deterministic acquisition of a mutex when the waiting thread is signaled. Thus, fairness and priority issues can be dealt with directly by the scheduling discipline. Furthermore, the current condition wait operation matches existing practice.

Scheduling Behavior of Mutexes and Condition Variables

Synchronization primitives that attempt to interfere with scheduling policy by specifying an ordering rule are considered undesirable. Threads waiting on mutexes and condition variables are selected to proceed in an order dependent upon the scheduling policy rather than in some fixed order (for example, FIFO or priority). Thus, the scheduling policy determines which thread(s) are awakened and allowed to proceed.

Timed Condition Wait

The *pthread_cond_timedwait*() function allows an application to give up waiting for a particular condition after a given amount of time. An example of its use follows:

```
(void) pthread_mutex_lock(&t.mn);
    t.waiters++;
    clock_gettime(CLOCK_REALTIME, &ts);
    ts.tv_sec += 5;
    rc = 0;
    while (! mypredicate(&t) && rc == 0)
        rc = pthread_cond_timedwait(&t.cond, &t.mn, &ts);
    t.waiters--;
    if (rc == 0 || mypredicate(&t))
        setmystate(&t);
(void) pthread_mutex_unlock(&t.mn);
```

By making the timeout parameter absolute, it does not need to be recomputed each time the program checks its blocking predicate. If the timeout was relative, it would have to be recomputed before each call. This would be especially difficult since such code would need to take into account the possibility of extra wakeups that result from extra broadcasts or signals on the condition variable that occur before either the predicate is true or the timeout is due.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_broadcast( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_destroy, pthread_condattr_init — destroy and initialize the condition variable attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_destroy(pthread_condattr_t *attr);
int pthread_condattr_init(pthread_condattr_t *attr);
```

DESCRIPTION

The *pthread_condattr_destroy*() function shall destroy a condition variable attributes object; the object becomes, in effect, uninitialized. An implementation may cause *pthread_condattr_destroy*() to set the object referenced by *attr* to an invalid value. A destroyed *attr* attributes object can be reinitialized using *pthread_condattr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_condattr_init*() function shall initialize a condition variable attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_condattr_init*() is called specifying an already initialized *attr* attributes object.

After a condition variable attributes object has been used to initialize one or more condition variables, any function affecting the attributes object (including destruction) shall not affect any previously initialized condition variables.

This volume of POSIX.1-2017 requires two attributes, the *clock* attribute and the *process-shared* attribute.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_destroy*() does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_destroy*() and *pthread_condattr_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the condition variable attributes object.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A *process-shared* attribute has been defined for condition variables for the same reason it has been defined for mutexes.

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_destroy*() does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

See also *pthread_attr_destroy*( ) and *pthread_mutex_destroy*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_attr_destroy( ), *pthread_cond_destroy*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_getclock, pthread_condattr_setclock — get and set the clock selection condition variable attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_getclock(const pthread_condattr_t *restrict attr,
    clockid_t *restrict clock_id);
int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

DESCRIPTION

The *pthread_condattr_getclock*() function shall obtain the value of the *clock* attribute from the attributes object referenced by *attr*.

The *pthread_condattr_setclock*() function shall set the *clock* attribute in an initialized attributes object referenced by *attr*. If *pthread_condattr_setclock*() is called with a *clock_id* argument that refers to a CPU-time clock, the call shall fail.

The *clock* attribute is the clock ID of the clock that shall be used to measure the timeout service of *pthread_cond_timedwait*(). The default value of the *clock* attribute shall refer to the system clock.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_getclock*() or *pthread_condattr_setclock*() does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_getclock*() function shall return zero and store the value of the clock attribute of *attr* into the object referenced by the *clock_id* argument. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_setclock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_setclock*() function may fail if:

EINVAL
The value specified by *clock_id* does not refer to a known clock, or is a CPU-time clock.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_getclock*() or *pthread_condattr_setclock*() does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_cond_timedwait*( ), *pthread_condattr_destroy*( ), *pthread_condattr_getpshared*( ), *pthread_create*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_getpshared, pthread_condattr_setpshared — get and set the process-shared condition variable attributes

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_getpshared(const pthread_condattr_t *restrict attr,
    int *restrict pshared);
int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_condattr_getpshared*() function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_condattr_setpshared*() function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a condition variable to be operated upon by any thread that has access to the memory where the condition variable is allocated, even if the condition variable is allocated in memory that is shared by multiple processes. See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements. The default value of the attribute is PTHREAD_PROCESS_PRIVATE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_condattr_getpshared*() or *pthread_condattr_setpshared*() does not refer to an initialized condition variable attributes object.

RETURN VALUE

If successful, the *pthread_condattr_setpshared*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_condattr_getpshared*() function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_condattr_setpshared*() function may fail if:

EINVAL
The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_condattr_getpshared*() or *pthread_condattr_setpshared*() does not refer to an initialized condition variable attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_cond_destroy*( ), *pthread_condattr_destroy*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_init — initialize the condition variable attributes object

SYNOPSIS

#include <pthread.h>

int pthread_condattr_init(pthread_condattr_t **attr*);

DESCRIPTION

Refer to *pthread_condattr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_setclock — set the clock selection condition variable attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_setclock(pthread_condattr_t *attr,
    clockid_t clock_id);
```

DESCRIPTION

Refer to *pthread_condattr_getclock*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_condattr_setpshared — set the process-shared condition variable attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_condattr_setpshared(pthread_condattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_condattr_getpshared*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_create — thread creation

SYNOPSIS

```
#include <pthread.h>

int pthread_create(pthread_t *restrict thread,
   const pthread_attr_t *restrict attr,
   void *(*start_routine)(void*), void *restrict arg);
```

DESCRIPTION

The *pthread_create*() function shall create a new thread, with attributes specified by *attr*, within a process. If *attr* is NULL, the default attributes shall be used. If the attributes specified by *attr* are modified later, the thread's attributes shall not be affected. Upon successful completion, *pthread_create*() shall store the ID of the created thread in the location referenced by *thread*.

The thread is created executing *start_routine* with *arg* as its sole argument. If the *start_routine* returns, the effect shall be as if there was an implicit call to *pthread_exit*() using the return value of *start_routine* as the exit status. Note that the thread in which *main*() was originally invoked differs from this. When it returns from *main*(), the effect shall be as if there was an implicit call to *exit*() using the return value of *main*() as the exit status.

The signal state of the new thread shall be initialized as follows:

* The signal mask shall be inherited from the creating thread.
* The set of signals pending for the new thread shall be empty.

The thread-local current locale and the alternate stack shall not be inherited.

The floating-point environment shall be inherited from the creating thread.

If *pthread_create*() fails, no new thread is created and the contents of the location referenced by *thread* are undefined.

If _POSIX_THREAD_CPUTIME is defined, the new thread shall have a CPU-time clock accessible, and the initial value of this clock shall be set to zero.

The behavior is undefined if the value specified by the *attr* argument to *pthread_create*() does not refer to an initialized thread attributes object.

RETURN VALUE

If successful, the *pthread_create*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_create*() function shall fail if:

EAGAIN
The system lacked the necessary resources to create another thread, or the system-imposed limit on the total number of threads in a process {PTHREAD_THREADS_MAX} would be exceeded.

EPERM
The caller does not have appropriate privileges to set the required scheduling parameters or scheduling policy.

The *pthread_create*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

There is no requirement on the implementation that the ID of the created thread be available before the newly created thread starts executing. The calling thread can obtain the ID of the created thread through the *thread* argument of the *pthread_create*() function, and the newly created thread can obtain its ID by a call to *pthread_self*().

RATIONALE

A suggested alternative to *pthread_create*() would be to define two separate operations: create and start. Some applications would find such behavior more natural. Ada, in particular, separates the ‘‘creation’’ of a task from its ‘‘activation’’.

Splitting the operation was rejected by the standard developers for many reasons:

* The number of calls required to start a thread would increase from one to two and thus place an additional burden on applications that do not require the additional synchronization. The second call, however, could be avoided by the additional complication of a start-up state attribute.

* An extra state would be introduced: ‘‘created but not started’’. This would require the standard to specify the behavior of the thread operations when the target has not yet started executing.

* For those applications that require such behavior, it is possible to simulate the two separate steps with the facilities that are currently provided. The *start_routine*() can synchronize by waiting on a condition variable that is signaled by the start operation.

An Ada implementor can choose to create the thread at either of two points in the Ada program: when the task object is created, or when the task is activated (generally at a ‘‘begin’’). If the first approach is adopted, the *start_routine*() needs to wait on a condition variable to receive the order to begin ‘‘activation’’. The second approach requires no such condition variable or extra synchronization. In either approach, a separate Ada task control block would need to be created when the task object is created to hold rendezvous queues, and so on.

An extension of the preceding model would be to allow the state of the thread to be modified between the create and start. This would allow the thread attributes object to be eliminated. This has been rejected because:

* All state in the thread attributes object has to be able to be set for the thread. This would require the definition of functions to modify thread attributes. There would be no reduction in the number of function calls required to set up the thread. In fact, for an application that creates all threads using identical attributes, the number of function calls required to set up the threads would be dramatically increased. Use of a thread attributes object permits the application to make one set of attribute setting function calls. Otherwise, the set of attribute setting function calls needs to be made for each thread creation.

* Depending on the implementation architecture, functions to set thread state would require kernel calls, or for other implementation reasons would not be able to be implemented as macros, thereby increasing the cost of thread creation.

* The ability for applications to segregate threads by class would be lost.

Another suggested alternative uses a model similar to that for process creation, such as ‘‘thread fork’’. The fork semantics would provide more flexibility and the ‘‘create’’ function can be implemented simply by doing a thread fork followed immediately by a call to the desired ‘‘start routine’’ for the thread. This alternative has these problems:

* For many implementations, the entire stack of the calling thread would need to be duplicated, since in many architectures there is no way to determine the size of the calling frame.

* Efficiency is reduced since at least some part of the stack has to be copied, even though in most cases the thread never needs the copied context, since it merely calls the desired start routine.

If an implementation detects that the value specified by the *attr* argument to *pthread_create*() does not refer to an initialized thread attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

fork( ), *pthread_exit*( ), *pthread_join*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_detach — detach a thread

SYNOPSIS

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

DESCRIPTION

The *pthread_detach*() function shall indicate to the implementation that storage for the thread *thread* can be reclaimed when that thread terminates. If *thread* has not terminated, *pthread_detach*() shall not cause it to terminate.

The behavior is undefined if the value specified by the *thread* argument to *pthread_detach*() does not refer to a joinable thread.

RETURN VALUE

If the call succeeds, *pthread_detach*() shall return 0; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_detach*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_join*() or *pthread_detach*() functions should eventually be called for every thread that is created so that storage associated with the thread may be reclaimed.

It has been suggested that a ''detach'' function is not necessary; the *detachstate* thread creation attribute is sufficient, since a thread need never be dynamically detached. However, need arises in at least two cases:

1. In a cancellation handler for a *pthread_join*() it is nearly essential to have a *pthread_detach*() function in order to detach the thread on which *pthread_join*() was waiting. Without it, it would be necessary to have the handler do another *pthread_join*() to attempt to detach the thread, which would both delay the cancellation processing for an unbounded period and introduce a new call to *pthread_join*(), which might itself need a cancellation handler. A dynamic detach is nearly essential in this case.
2. In order to detach the ''initial thread'' (as may be desirable in processes that set up server threads).

If an implementation detects that the value specified by the *thread* argument to *pthread_detach*() does not refer to a joinable thread, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_join( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_equal — compare thread IDs

SYNOPSIS

```
#include <pthread.h>

int pthread_equal(pthread_t t1, pthread_t t2);
```

DESCRIPTION

This function shall compare the thread IDs *t1* and *t2*.

RETURN VALUE

The *pthread_equal*() function shall return a non-zero value if *t1* and *t2* are equal; otherwise, zero shall be returned.

If either *t1* or *t2* are not valid thread IDs, the behavior is undefined.

ERRORS

No errors are defined.

The *pthread_equal*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Implementations may choose to define a thread ID as a structure. This allows additional flexibility and robustness over using an **int**. For example, a thread ID could include a sequence number that allows detection of "dangling IDs" (copies of a thread ID that has been detached). Since the C language does not support comparison on structure types, the *pthread_equal*() function is provided to compare thread IDs.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_self*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_exit — thread termination

SYNOPSIS

```
#include <pthread.h>

void pthread_exit(void *value_ptr);
```

DESCRIPTION

The *pthread_exit*() function shall terminate the calling thread and make the value *value_ptr* available to any successful join with the terminating thread. Any cancellation cleanup handlers that have been pushed and not yet popped shall be popped in the reverse order that they were pushed and then executed. After all cancellation cleanup handlers have been executed, if the thread has any thread-specific data, appropriate destructor functions shall be called in an unspecified order. Thread termination does not release any application visible process resources, including, but not limited to, mutexes and file descriptors, nor does it perform any process-level cleanup actions, including, but not limited to, calling any *atexit*() routines that may exist.

An implicit call to *pthread_exit*() is made when a thread other than the thread in which *main*() was first invoked returns from the start routine that was used to create it. The function's return value shall serve as the thread's exit status.

The behavior of *pthread_exit*() is undefined if called from a cancellation cleanup handler or destructor function that was invoked as a result of either an implicit or explicit call to *pthread_exit*().

After a thread has terminated, the result of access to local (auto) variables of the thread is undefined. Thus, references to local variables of the exiting thread should not be used for the *pthread_exit*() *value_ptr* parameter value.

The process shall exit with an exit status of 0 after the last thread has been terminated. The behavior shall be as if the implementation called *exit*() with a zero argument at thread termination time.

RETURN VALUE

The *pthread_exit*() function cannot return to its caller.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The normal mechanism by which a thread terminates is to return from the routine that was specified in the *pthread_create*() call that started it. The *pthread_exit*() function provides the capability for a thread to terminate without requiring a return from the start routine of that thread, thereby providing a function analogous to *exit*().

Regardless of the method of thread termination, any cancellation cleanup handlers that have been pushed and not yet popped are executed, and the destructors for any existing thread-specific data are executed. This volume of POSIX.1-2017 requires that cancellation cleanup handlers be popped and called in order. After all cancellation cleanup handlers have been executed, thread-specific data destructors are called, in an unspecified order, for each item of thread-specific data that exists in the thread. This ordering is necessary because cancellation cleanup handlers may rely on thread-specific data.

As the meaning of the status is determined by the application (except when the thread has been canceled, in which case it is PTHREAD_CANCELED), the implementation has no idea what an illegal status value is, which is why no address error checking is done.

FUTURE DIRECTIONS

None.

SEE ALSO

exit( ), *pthread_create*( ), *pthread_join*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_getconcurrency, pthread_setconcurrency — get and set the level of concurrency

SYNOPSIS

```
#include <pthread.h>

int pthread_getconcurrency(void);
int pthread_setconcurrency(int new_level);
```

DESCRIPTION

Unbound threads in a process may or may not be required to be simultaneously active. By default, the threads implementation ensures that a sufficient number of threads are active so that the process can continue to make progress. While this conserves system resources, it may not produce the most effective level of concurrency.

The *pthread_setconcurrency*() function allows an application to inform the threads implementation of its desired concurrency level, *new_level*. The actual level of concurrency provided by the implementation as a result of this function call is unspecified.

If *new_level* is zero, it causes the implementation to maintain the concurrency level at its discretion as if *pthread_setconcurrency*() had never been called.

The *pthread_getconcurrency*() function shall return the value set by a previous call to the *pthread_setconcurrency*() function. If the *pthread_setconcurrency*() function was not previously called, this function shall return zero to indicate that the implementation is maintaining the concurrency level.

A call to *pthread_setconcurrency*() shall inform the implementation of its desired concurrency level. The implementation shall use this as a hint, not a requirement.

If an implementation does not support multiplexing of user threads on top of several kernel-scheduled entities, the *pthread_setconcurrency*() and *pthread_getconcurrency*() functions are provided for source code compatibility but they shall have no effect when called. To maintain the function semantics, the *new_level* parameter is saved when *pthread_setconcurrency*() is called so that a subsequent call to *pthread_getconcurrency*() shall return the same value.

RETURN VALUE

If successful, the *pthread_setconcurrency*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_getconcurrency*() function shall always return the concurrency level set by a previous call to *pthread_setconcurrency*(). If the *pthread_setconcurrency*() function has never been called, *pthread_getconcurrency*() shall return zero.

ERRORS

The *pthread_setconcurrency*() function shall fail if:

EINVAL
The value specified by *new_level* is negative.

EAGAIN
The value specified by *new_level* would cause a system resource to be exceeded.

The *pthread_setconcurrency*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Application developers should note that an implementation can always ignore any calls to *pthread_setconcurrency*() and return a constant for *pthread_getconcurrency*(). For this reason, it is not recommended that portable applications use this function.

RATIONALE

None.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_getcpuclockid — access a thread CPU-time clock (**ADVANCED REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_getcpuclockid(pthread_t thread_id, clockid_t *clock_id);
```

DESCRIPTION

The *pthread_getcpuclockid*() function shall return in *clock_id* the clock ID of the CPU-time clock of the thread specified by *thread_id*, if the thread specified by *thread_id* exists.

RETURN VALUE

Upon successful completion, *pthread_getcpuclockid*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_getcpuclockid*() function is part of the Thread CPU-Time Clocks option and need not be provided on all implementations.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getcpuclockid( ), *clock_getres*( ), *timer_create*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_getschedparam, pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_getschedparam(pthread_t thread, int *restrict policy,
   struct sched_param *restrict param);
int pthread_setschedparam(pthread_t thread, int policy,
   const struct sched_param *param);
```

DESCRIPTION

The *pthread_getschedparam*() and *pthread_setschedparam*() functions shall, respectively, get and set the scheduling policy and parameters of individual threads within a multi-threaded process to be retrieved and set. For SCHED_FIFO and SCHED_RR, the only required member of the **sched_param** structure is the priority *sched_priority*. For SCHED_OTHER, the affected scheduling parameters are implementation-defined.

The *pthread_getschedparam*() function shall retrieve the scheduling policy and scheduling parameters for the thread whose thread ID is given by *thread* and shall store those values in *policy* and *param*, respectively. The priority value returned from *pthread_getschedparam*() shall be the value specified by the most recent *pthread_setschedparam*(), *pthread_setschedprio*(), or *pthread_create*() call affecting the target thread. It shall not reflect any temporary adjustments to its priority as a result of any priority inheritance or ceiling functions. The *pthread_setschedparam*() function shall set the scheduling policy and associated scheduling parameters for the thread whose thread ID is given by *thread* to the policy and associated parameters provided in *policy* and *param*, respectively.

The *policy* parameter may have the value SCHED_OTHER, SCHED_FIFO, or SCHED_RR. The scheduling parameters for the SCHED_OTHER policy are implementation-defined. The SCHED_FIFO and SCHED_RR policies shall have a single scheduling parameter, *priority*.

If _POSIX_THREAD_SPORADIC_SERVER is defined, then the *policy* argument may have the value SCHED_SPORADIC, with the exception for the *pthread_setschedparam*() function that if the scheduling policy was not SCHED_SPORADIC at the time of the call, it is implementation-defined whether the function is supported; in other words, the implementation need not allow the application to dynamically change the scheduling policy to SCHED_SPORADIC. The sporadic server scheduling policy has the associated parameters *sched_ss_low_priority*, *sched_ss_repl_period*, *sched_ss_init_budget*, *sched_priority*, and *sched_ss_max_repl*. The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail. The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

If the *pthread_setschedparam*() function fails, the scheduling parameters shall not be changed for the target thread.

RETURN VALUE

If successful, the *pthread_getschedparam*() and *pthread_setschedparam*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setschedparam*() function shall fail if:

ENOTSUP
An attempt was made to set the policy or scheduling parameters to an unsupported value.

ENOTSUP
An attempt was made to dynamically change the scheduling policy to SCHED_SPORADIC, and the implementation does not support this change.

The *pthread_setschedparam*() function may fail if:

EINVAL
The value specified by *policy* or one of the scheduling parameters associated with the scheduling policy *policy* is invalid.

EPERM
The caller does not have appropriate privileges to set either the scheduling parameters or the scheduling policy of the specified thread.

EPERM
The implementation does not allow the application to modify one of the parameters to the value specified.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_setschedprio( ), *sched_getparam*( ), *sched_getscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_getspecific, pthread_setspecific — thread-specific data management

SYNOPSIS

```
#include <pthread.h>

void *pthread_getspecific(pthread_key_t key);
int pthread_setspecific(pthread_key_t key, const void *value);
```

DESCRIPTION

The *pthread_getspecific*() function shall return the value currently bound to the specified *key* on behalf of the calling thread.

The *pthread_setspecific*() function shall associate a thread-specific *value* with a *key* obtained via a previous call to *pthread_key_create*(). Different threads may bind different values to the same key. These values are typically pointers to blocks of dynamically allocated memory that have been reserved for use by the calling thread.

The effect of calling *pthread_getspecific*() or *pthread_setspecific*() with a *key* value not obtained from *pthread_key_create*() or after *key* has been deleted with *pthread_key_delete*() is undefined.

Both *pthread_getspecific*() and *pthread_setspecific*() may be called from a thread-specific data destructor function. A call to *pthread_getspecific*() for the thread-specific data key being destroyed shall return the value NULL, unless the value is changed (after the destructor starts) by a call to *pthread_setspecific*(). Calling *pthread_setspecific*() from a thread-specific data destructor routine may result either in lost storage (after at least PTHREAD_DESTRUCTOR_ITERATIONS attempts at destruction) or in an infinite loop.

Both functions may be implemented as macros.

RETURN VALUE

The *pthread_getspecific*() function shall return the thread-specific data value associated with the given *key*. If no thread-specific data value is associated with *key*, then the value NULL shall be returned.

If successful, the *pthread_setspecific*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

No errors are returned from *pthread_getspecific*().

The *pthread_setspecific*() function shall fail if:

ENOMEM
Insufficient memory exists to associate the non-NULL value with the key.

The *pthread_setspecific*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Performance and ease-of-use of *pthread_getspecific*() are critical for functions that rely on maintaining state in thread-specific data. Since no errors are required to be detected by it, and since the only error that could be detected is the use of an invalid key, the function to *pthread_getspecific*() has been designed to favor speed and simplicity over error reporting.

If an implementation detects that the value specified by the *key* argument to *pthread_setspecific*() does not refer to a a key value obtained from *pthread_key_create*() or refers to a key that has been deleted with *pthread_key_delete*(), it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_join — wait for thread termination

SYNOPSIS

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **value_ptr);
```

DESCRIPTION

The *pthread_join*() function shall suspend execution of the calling thread until the target *thread* terminates, unless the target *thread* has already terminated. On return from a successful *pthread_join*() call with a non-NULL *value_ptr* argument, the value passed to *pthread_exit*() by the terminating thread shall be made available in the location referenced by *value_ptr*. When a *pthread_join*() returns successfully, the target thread has been terminated. The results of multiple simultaneous calls to *pthread_join*() specifying the same target thread are undefined. If the thread calling *pthread_join*() is canceled, then the target thread shall not be detached.

It is unspecified whether a thread that has exited but remains unjoined counts against {PTHREAD_THREADS_MAX}.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join*() does not refer to a joinable thread.

The behavior is undefined if the value specified by the *thread* argument to *pthread_join*() refers to the calling thread.

RETURN VALUE

If successful, the *pthread_join*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_join*() function may fail if:

EDEADLK
A deadlock was detected.

The *pthread_join*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

An example of thread creation and deletion follows:

```
typedef struct {
    int *ar;
    long n;
} subarray;

void *
incer(void *arg)
{
    long i;

    for (i = 0; i < ((subarray *)arg)->n; i++)
        ((subarray *)arg)->ar[i]++;
}

int main(void)
```

```
{
    int        ar[1000000];
    pthread_t  th1, th2;
    subarray   sb1, sb2;

    sb1.ar = &ar[0];
    sb1.n  = 500000;
    (void) pthread_create(&th1, NULL, incer, &sb1);

    sb2.ar = &ar[500000];
    sb2.n  = 500000;
    (void) pthread_create(&th2, NULL, incer, &sb2);

    (void) pthread_join(th1, NULL);
    (void) pthread_join(th2, NULL);
    return 0;
}
```

APPLICATION USAGE

None.

RATIONALE

The *pthread_join*() function is a convenience that has proven useful in multi-threaded applications. It is true that a programmer could simulate this function if it were not provided by passing extra state as part of the argument to the *start_routine*(). The terminating thread would set a flag to indicate termination and broadcast a condition that is part of that state; a joining thread would wait on that condition variable. While such a technique would allow a thread to wait on more complex conditions (for example, waiting for multiple threads to terminate), waiting on individual thread termination is considered widely useful. Also, including the *pthread_join*() function in no way precludes a programmer from coding such complex waits. Thus, while not a primitive, including *pthread_join*() in this volume of POSIX.1-2017 was considered valuable.

The *pthread_join*() function provides a simple mechanism allowing an application to wait for a thread to terminate. After the thread terminates, the application may then choose to clean up resources that were used by the thread. For instance, after *pthread_join*() returns, any application-provided stack storage could be reclaimed.

The *pthread_join*() or *pthread_detach*() function should eventually be called for every thread that is created with the *detachstate* attribute set to PTHREAD_CREATE_JOINABLE so that storage associated with the thread may be reclaimed.

The interaction between *pthread_join*() and cancellation is well-defined for the following reasons:

* The *pthread_join*() function, like all other non-async-cancel-safe functions, can only be called with deferred cancelability type.
* Cancellation cannot occur in the disabled cancelability state.

Thus, only the default cancelability state need be considered. As specified, either the *pthread_join*() call is canceled, or it succeeds, but not both. The difference is obvious to the application, since either a cancellation handler is run or *pthread_join*() returns. There are no race conditions since *pthread_join*() was called in the deferred cancelability state.

If an implementation detects that the value specified by the *thread* argument to *pthread_join*() does not refer to a joinable thread, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *thread* argument to *pthread_join*() refers to the calling thread, it is recommended that the function should fail and report an **[EDEADLK]** error.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_key_create — thread-specific data key creation

SYNOPSIS

#include <pthread.h>

int pthread_key_create(pthread_key_t **key*, void (**destructor*)(void*));

DESCRIPTION

The *pthread_key_create*() function shall create a thread-specific data key visible to all threads in the process. Key values provided by *pthread_key_create*() are opaque objects used to locate thread-specific data. Although the same key value may be used by different threads, the values bound to the key by *pthread_setspecific*() are maintained on a per-thread basis and persist for the life of the calling thread.

Upon key creation, the value NULL shall be associated with the new key in all active threads. Upon thread creation, the value NULL shall be associated with all defined keys in the new thread.

An optional destructor function may be associated with each key value. At thread exit, if a key value has a non-NULL destructor pointer, and the thread has a non-NULL value associated with that key, the value of the key is set to NULL, and then the function pointed to is called with the previously associated value as its sole argument. The order of destructor calls is unspecified if more than one destructor exists for a thread when it exits.

If, after all the destructors have been called for all non-NULL values with associated destructors, there are still some non-NULL values with associated destructors, then the process is repeated. If, after at least {PTHREAD_DESTRUCTOR_ITERATIONS} iterations of destructor calls for outstanding non-NULL values, there are still some non-NULL values with associated destructors, implementations may stop calling destructors, or they may continue calling destructors until no non-NULL values with associated destructors exist, even though this might result in an infinite loop.

RETURN VALUE

If successful, the *pthread_key_create*() function shall store the newly created key value at **key* and shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_create*() function shall fail if:

EAGAIN
The system lacked the necessary resources to create another thread-specific data key, or the system-imposed limit on the total number of keys per process {PTHREAD_KEYS_MAX} has been exceeded.

ENOMEM
Insufficient memory exists to create the key.

The *pthread_key_create*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

The following example demonstrates a function that initializes a thread-specific data key when it is first called, and associates a thread-specific object with each calling thread, initializing this object when necessary.

```
static pthread_key_t key;
static pthread_once_t key_once = PTHREAD_ONCE_INIT;
static void
make_key()
{
    (void) pthread_key_create(&key, NULL);
}

func()
{
    void *ptr;

    (void) pthread_once(&key_once, make_key);
    if ((ptr = pthread_getspecific(key)) == NULL) {
        ptr = malloc(OBJECT_SIZE);
        ...
        (void) pthread_setspecific(key, ptr);
    }
    ...
}
```

Note that the key has to be initialized before *pthread_getspecific*() or *pthread_setspecific*() can be used. The *pthread_key_create*() call could either be explicitly made in a module initialization routine, or it can be done implicitly by the first call to a module as in this example. Any attempt to use the key before it is initialized is a programming error, making the code below incorrect.

```
static pthread_key_t key;

func()
{
    void *ptr;

   /* KEY NOT INITIALIZED!!!  THIS WILL NOT WORK!!! */
    if ((ptr = pthread_getspecific(key)) == NULL &&
        pthread_setspecific(key, NULL) != 0) {
        pthread_key_create(&key, NULL);
        ...
    }
}
```

APPLICATION USAGE

None.

RATIONALE

Destructor Functions

Normally, the value bound to a key on behalf of a particular thread is a pointer to storage allocated dynamically on behalf of the calling thread. The destructor functions specified with *pthread_key_create*() are intended to be used to free this storage when the thread exits. Thread cancellation cleanup handlers cannot be used for this purpose because thread-specific data may persist outside the lexical scope in which the cancellation cleanup handlers operate.

If the value associated with a key needs to be updated during the lifetime of the thread, it may be necessary to release the storage associated with the old value before the new value is bound. Although the *pthread_setspecific*() function could do this automatically, this feature is not needed often enough to justify the added complexity. Instead, the programmer is responsible for freeing the stale storage:

```
pthread_getspecific(key, &old);
new = allocate();
destructor(old);
```

```
pthread_setspecific(key, new);
```

Note: The above example could leak storage if run with asynchronous cancellation enabled. No such problems occur in the default cancellation state if no cancellation points occur between the get and set.

There is no notion of a destructor-safe function. If an application does not call *pthread_exit*() from a signal handler, or if it blocks any signal whose handler may call *pthread_exit*() while calling async-unsafe functions, all functions may be safely called from destructors.

Non-Idempotent Data Key Creation

There were requests to make *pthread_key_create*() idempotent with respect to a given *key* address parameter. This would allow applications to call *pthread_key_create*() multiple times for a given *key* address and be guaranteed that only one key would be created. Doing so would require the key value to be previously initialized (possibly at compile time) to a known null value and would require that implicit mutual-exclusion be performed based on the address and contents of the *key* parameter in order to guarantee that exactly one key would be created.

Unfortunately, the implicit mutual-exclusion would not be limited to only *pthread_key_create*(). On many implementations, implicit mutual-exclusion would also have to be performed by *pthread_getspecific*() and *pthread_setspecific*() in order to guard against using incompletely stored or not-yet-visible key values. This could significantly increase the cost of important operations, particularly *pthread_getspecific*().

Thus, this proposal was rejected. The *pthread_key_create*() function performs no implicit synchronization. It is the responsibility of the programmer to ensure that it is called exactly once per key before use of the key. Several straightforward mechanisms can already be used to accomplish this, including calling explicit module initialization functions, using mutexes, and using *pthread_once*(). This places no significant burden on the programmer, introduces no possibly confusing *ad hoc* implicit synchronization mechanism, and potentially allows commonly used thread-specific data operations to be more efficient.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_getspecific( ), *pthread_key_delete*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_key_delete — thread-specific data key deletion

SYNOPSIS

#include <pthread.h>

int pthread_key_delete(pthread_key_t *key*);

DESCRIPTION

The *pthread_key_delete*() function shall delete a thread-specific data key previously returned by *pthread_key_create*(). The thread-specific data values associated with *key* need not be NULL at the time *pthread_key_delete*() is called. It is the responsibility of the application to free any application storage or perform any cleanup actions for data structures related to the deleted key or associated thread-specific data in any threads; this cleanup can be done either before or after *pthread_key_delete*() is called. Any attempt to use *key* following the call to *pthread_key_delete*() results in undefined behavior.

The *pthread_key_delete*() function shall be callable from within destructor functions. No destructor functions shall be invoked by *pthread_key_delete*(). Any destructor function that may have been associated with *key* shall no longer be called upon thread exit.

RETURN VALUE

If successful, the *pthread_key_delete*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_key_delete*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

A thread-specific data key deletion function has been included in order to allow the resources associated with an unused thread-specific data key to be freed. Unused thread-specific data keys can arise, among other scenarios, when a dynamically loaded module that allocated a key is unloaded.

Conforming applications are responsible for performing any cleanup actions needed for data structures associated with the key to be deleted, including data referenced by thread-specific data values. No such cleanup is done by *pthread_key_delete*(). In particular, destructor functions are not called. There are several reasons for this division of responsibility:

1. The associated destructor functions used to free thread-specific data at thread exit time are only guaranteed to work correctly when called in the thread that allocated the thread-specific data. (Destructors themselves may utilize thread-specific data.) Thus, they cannot be used to free thread-specific data in other threads at key deletion time. Attempting to have them called by other threads at key deletion time would require other threads to be asynchronously interrupted. But since interrupted threads could be in an arbitrary state, including holding locks necessary for the destructor to run, this approach would fail. In general, there is no safe mechanism whereby an implementation could free thread-specific data at key deletion time.

2. Even if there were a means of safely freeing thread-specific data associated with keys to be deleted, doing so would require that implementations be able to enumerate the threads with non-NULL data and potentially keep them from creating more thread-specific data while the key deletion is occurring. This special case could cause extra synchronization in the normal case, which would otherwise be

unnecessary.

For an application to know that it is safe to delete a key, it has to know that all the threads that might potentially ever use the key do not attempt to use it again. For example, it could know this if all the client threads have called a cleanup procedure declaring that they are through with the module that is being shut down, perhaps by setting a reference count to zero.

If an implementation detects that the value specified by the *key* argument to *pthread_key_delete*() does not refer to a a key value obtained from *pthread_key_create*() or refers to a key that has been deleted with *pthread_key_delete*(), it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_key_create( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_kill — send a signal to a thread

SYNOPSIS

#include <signal.h>

int pthread_kill(pthread_t *thread*, int *sig*);

DESCRIPTION

The *pthread_kill*() function shall request that a signal be delivered to the specified thread.

As in *kill*(), if *sig* is zero, error checking shall be performed but no signal shall actually be sent.

RETURN VALUE

Upon successful completion, the function shall return a value of zero. Otherwise, the function shall return an error number. If the *pthread_kill*() function fails, no signal shall be sent.

ERRORS

The *pthread_kill*() function shall fail if:

EINVAL
The value of the *sig* argument is an invalid or unsupported signal number.

The *pthread_kill*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_kill*() function provides a mechanism for asynchronously directing a signal at a thread in the calling process. This could be used, for example, by one thread to affect broadcast delivery of a signal to a set of threads.

Note that *pthread_kill*() only causes the signal to be handled in the context of the given thread; the signal action (termination or stopping) affects the process as a whole.

RATIONALE

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

Existing implementations vary on the result of a *pthread_kill*() with a thread ID indicating an inactive thread (a terminated thread that has not been detached or joined). Some indicate success on such a call, while others give an error of **[ESRCH]**. Since the definition of thread lifetime in this volume of POSIX.1-2017 covers inactive threads, the **[ESRCH]** error as described is inappropriate in this case. In particular, this means that an application cannot have one thread check for termination of another with *pthread_kill*().

FUTURE DIRECTIONS

A future version of this standard may require that *pthread_kill*() not fail with **[ESRCH]** in the case of sending signals to an inactive thread (a terminated thread not yet detached or joined), even though no signal will be delivered because the thread is no longer running.

SEE ALSO

kill( ), *pthread_self*( ), *raise*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_consistent — mark state protected by robust mutex as consistent

SYNOPSIS

#include <pthread.h>

int pthread_mutex_consistent(pthread_mutex_t **mutex*);

DESCRIPTION

If *mutex* is a robust mutex in an inconsistent state, the *pthread_mutex_consistent*() function can be used to mark the state protected by the mutex referenced by *mutex* as consistent again.

If an owner of a robust mutex terminates while holding the mutex, the mutex becomes inconsistent and the next thread that acquires the mutex lock shall be notified of the state by the return value **[EOWNERDEAD]**. In this case, the mutex does not become normally usable again until the state is marked consistent.

If the thread which acquired the mutex lock with the return value **[EOWNERDEAD]** terminates before calling either *pthread_mutex_consistent*() or *pthread_mutex_unlock*(), the next thread that acquires the mutex lock shall be notified about the state of the mutex by the return value **[EOWNERDEAD]**.

The behavior is undefined if the value specified by the *mutex* argument to *pthread_mutex_consistent*() does not refer to an initialized mutex.

RETURN VALUE

Upon successful completion, the *pthread_mutex_consistent*() function shall return zero. Otherwise, an error value shall be returned to indicate the error.

ERRORS

The *pthread_mutex_consistent*() function shall fail if:

EINVAL
The mutex object referenced by *mutex* is not robust or does not protect an inconsistent state.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *pthread_mutex_consistent*() function is only responsible for notifying the implementation that the state protected by the mutex has been recovered and that normal operations with the mutex can be resumed. It is the responsibility of the application to recover the state so it can be reused. If the application is not able to perform the recovery, it can notify the implementation that the situation is unrecoverable by a call to *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*(), in which case subsequent threads that attempt to lock the mutex will fail to acquire the lock and be returned **[ENOTRECOVERABLE]**.

RATIONALE

If an implementation detects that the value specified by the *mutex* argument to *pthread_mutex_consistent*() does not refer to an initialized mutex, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_lock( ), *pthread_mutexattr_getrobust*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_destroy, pthread_mutex_init — destroy and initialize a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_destroy(pthread_mutex_t *mutex);
int pthread_mutex_init(pthread_mutex_t *restrict mutex,
   const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

DESCRIPTION

The *pthread_mutex_destroy*() function shall destroy the mutex object referenced by *mutex*; the mutex object becomes, in effect, uninitialized. An implementation may cause *pthread_mutex_destroy*() to set the object referenced by *mutex* to an invalid value.

A destroyed mutex object can be reinitialized using *pthread_mutex_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

It shall be safe to destroy an initialized mutex that is unlocked. Attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *pthread_cond_timedwait*() or *pthread_cond_wait*() call by another thread, results in undefined behavior.

The *pthread_mutex_init*() function shall initialize the mutex referenced by *mutex* with attributes specified by *attr*. If *attr* is NULL, the default mutex attributes are used; the effect shall be the same as passing the address of a default mutex attributes object. Upon successful initialization, the state of the mutex becomes initialized and unlocked.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

Attempting to initialize an already initialized mutex results in undefined behavior.

In cases where default mutex attributes are appropriate, the macro PTHREAD_MUTEX_INITIALIZER can be used to initialize mutexes. The effect shall be equivalent to dynamic initialization by a call to *pthread_mutex_init*() with parameter *attr* specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *mutex* argument to *pthread_mutex_destroy*() does not refer to an initialized mutex.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutex_init*() does not refer to an initialized mutex attributes object.

RETURN VALUE

If successful, the *pthread_mutex_destroy*() and *pthread_mutex_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_init*() function shall fail if:

EAGAIN
The system lacked the necessary resources (other than memory) to initialize another mutex.

ENOMEM
Insufficient memory exists to initialize the mutex.

EPERM
The caller does not have the privilege to perform the operation.

The *pthread_mutex_init*() function may fail if:

EINVAL
The attributes object referenced by *attr* has the robust mutex attribute set without the process-shared attribute being set.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
If an implementation detects that the value specified by the *mutex* argument to *pthread_mutex_destroy*() does not refer to an initialized mutex, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *mutex* argument to *pthread_mutex_destroy*() or *pthread_mutex_init*() refers to a locked mutex or a mutex that is referenced (for example, while being used in a *pthread_cond_timedwait*() or *pthread_cond_wait*()) by another thread, or detects that the value specified by the *mutex* argument to *pthread_mutex_init*() refers to an already initialized mutex, it is recommended that the function should fail and report an **[EBUSY]** error.

If an implementation detects that the value specified by the *attr* argument to *pthread_mutex_init*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

Alternate Implementations Possible
This volume of POSIX.1-2017 supports several alternative implementations of mutexes. An implementation may store the lock directly in the object of type **pthread_mutex_t**. Alternatively, an implementation may store the lock in the heap and merely store a pointer, handle, or unique ID in the mutex object. Either implementation has advantages or may be required on certain hardware configurations. So that portable code can be written that is invariant to this choice, this volume of POSIX.1-2017 does not define assignment or equality for this type, and it uses the term ‘‘initialize’’ to reinforce the (more restrictive) notion that the lock may actually reside in the mutex object itself.

Note that this precludes an over-specification of the type of the mutex or condition variable and motivates the opaqueness of the type.

An implementation is permitted, but not required, to have *pthread_mutex_destroy*() store an illegal value into the mutex. This may help detect erroneous programs that try to lock (or otherwise reference) a mutex that has already been destroyed.

Tradeoff Between Error Checks and Performance Supported
Many error conditions that can occur are not required to be detected by the implementation in order to let implementations trade off performance *versus* degree of error checking according to the needs of their specific applications and execution environment. As a general rule, conditions caused by the system (such as insufficient memory) are required to be detected, but conditions caused by an erroneously coded application (such as failing to provide adequate synchronization to prevent a mutex from being deleted while in use) are specified to result in undefined behavior.

A wide range of implementations is thus made possible. For example, an implementation intended for application debugging may implement all of the error checks, but an implementation running a single, provably correct application under very tight performance constraints in an embedded computer might implement minimal checks. An implementation might even be provided in two versions, similar to the options that compilers provide: a full-checking, but slower version; and a limited-checking, but faster version. To forbid this optionality would be a disservice to users.

By carefully limiting the use of ‘‘undefined behavior’’ only to things that an erroneous (badly coded) application might do, and by defining that resource-not-available errors are mandatory, this volume of

POSIX.1-2017 ensures that a fully-conforming application is portable across the full range of implementations, while not forcing all implementations to add overhead to check for numerous things that a correct program never does. When the behavior is undefined, no error number is specified to be returned on implementations that do detect the condition. This is because undefined behavior means *anything* can happen, which includes returning with any value (which might happen to be a valid, but different, error number). However, since the error number might be useful to application developers when diagnosing problems during application development, a recommendation is made in rationale that implementors should return a particular error number if their implementation does detect the condition.

Why No Limits are Defined

Defining symbols for the maximum number of mutexes and condition variables was considered but rejected because the number of these objects may change dynamically. Furthermore, many implementations place these objects into application memory; thus, there is no explicit maximum.

Static Initializers for Mutexes and Condition Variables

Providing for static initialization of statically allocated synchronization objects allows modules with private static synchronization variables to avoid runtime initialization tests and overhead. Furthermore, it simplifies the coding of self-initializing modules. Such modules are common in C libraries, where for various reasons the design calls for self-initialization instead of requiring an explicit module initialization function to be called. An example use of static initialization follows.

Without static initialization, a self-initializing routine *foo*() might look as follows:

```
static pthread_once_t foo_once = PTHREAD_ONCE_INIT;
static pthread_mutex_t foo_mutex;

void foo_init()
{
    pthread_mutex_init(&foo_mutex, NULL);
}

void foo()
{
    pthread_once(&foo_once, foo_init);
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

With static initialization, the same routine could be coded as follows:

```
static pthread_mutex_t foo_mutex = PTHREAD_MUTEX_INITIALIZER;

void foo()
{
    pthread_mutex_lock(&foo_mutex);
   /* Do work. */
    pthread_mutex_unlock(&foo_mutex);
}
```

Note that the static initialization both eliminates the need for the initialization test inside *pthread_once*() and the fetch of *&foo_mutex* to learn the address to be passed to *pthread_mutex_lock*() or *pthread_mutex_unlock*().

Thus, the C code written to initialize static objects is simpler on all systems and is also faster on a large class of systems; those where the (entire) synchronization object can be stored in application memory.

Yet the locking performance question is likely to be raised for machines that require mutexes to be allocated out of special memory. Such machines actually have to have mutexes and possibly condition variables contain pointers to the actual hardware locks. For static initialization to work on such machines, *pthread_mutex_lock*() also has to test whether or not the pointer to the actual lock has been allocated. If it has not, *pthread_mutex_lock*() has to initialize it before use. The reservation of such resources can be made when the program is loaded, and hence return codes have not been added to mutex locking and condition variable waiting to indicate failure to complete initialization.

This runtime test in *pthread_mutex_lock*() would at first seem to be extra work; an extra test is required to see whether the pointer has been initialized. On most machines this would actually be implemented as a fetch of the pointer, testing the pointer against zero, and then using the pointer if it has already been initialized. While the test might seem to add extra work, the extra effort of testing a register is usually negligible since no extra memory references are actually done. As more and more machines provide caches, the real expenses are memory references, not instructions executed.

Alternatively, depending on the machine architecture, there are often ways to eliminate *all* overhead in the most important case: on the lock operations that occur *after* the lock has been initialized. This can be done by shifting more overhead to the less frequent operation: initialization. Since out-of-line mutex allocation also means that an address has to be dereferenced to find the actual lock, one technique that is widely applicable is to have static initialization store a bogus value for that address; in particular, an address that causes a machine fault to occur. When such a fault occurs upon the first attempt to lock such a mutex, validity checks can be done, and then the correct address for the actual lock can be filled in. Subsequent lock operations incur no extra overhead since they do not ''fault''. This is merely one technique that can be used to support static initialization, while not adversely affecting the performance of lock acquisition. No doubt there are other techniques that are highly machine-dependent.

The locking overhead for machines doing out-of-line mutex allocation is thus similar for modules being implicitly initialized, where it is improved for those doing mutex allocation entirely inline. The inline case is thus made much faster, and the out-of-line case is not significantly worse.

Besides the issue of locking performance for such machines, a concern is raised that it is possible that threads would serialize contending for initialization locks when attempting to finish initializing statically allocated mutexes. (Such finishing would typically involve taking an internal lock, allocating a structure, storing a pointer to the structure in the mutex, and releasing the internal lock.) First, many implementations would reduce such serialization by hashing on the mutex address. Second, such serialization can only occur a bounded number of times. In particular, it can happen at most as many times as there are statically allocated synchronization objects. Dynamically allocated objects would still be initialized via *pthread_mutex_init*() or *pthread_cond_init*().

Finally, if none of the above optimization techniques for out-of-line allocation yields sufficient performance for an application on some implementation, the application can avoid static initialization altogether by explicitly initializing all synchronization objects with the corresponding *pthread_*_init*( ) functions, which are supported by all implementations. An implementation can also document the tradeoffs and advise which initialization technique is more efficient for that particular implementation.

Destroying Mutexes

A mutex can be destroyed immediately after it is unlocked. However, since attempting to destroy a locked mutex, or a mutex that another thread is attempting to lock, or a mutex that is being used in a *pthread_cond_timedwait*() or *pthread_cond_wait*() call by another thread, results in undefined behavior, care must be taken to ensure that no other thread may be referencing the mutex.

Robust Mutexes

Implementations are required to provide robust mutexes for mutexes with the process-shared attribute set to PTHREAD_PROCESS_SHARED. Implementations are allowed, but not required, to provide robust mutexes when the process-shared attribute is set to PTHREAD_PROCESS_PRIVATE.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_getprioceiling( ), *pthread_mutexattr_getrobust*( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( ), *pthread_mutexattr_getpshared*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_getprioceiling, pthread_mutex_setprioceiling — get and set the priority ceiling of a mutex (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_getprioceiling(const pthread_mutex_t *restrict mutex,
    int *restrict prioceiling);
int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

DESCRIPTION

The *pthread_mutex_getprioceiling*() function shall return the current priority ceiling of the mutex.

The *pthread_mutex_setprioceiling*() function shall attempt to lock the mutex as if by a call to *pthread_mutex_lock*(), except that the process of locking the mutex need not adhere to the priority protect protocol. On acquiring the mutex it shall change the mutex's priority ceiling and then release the mutex as if by a call to *pthread_mutex_unlock*(). When the change is successful, the previous value of the priority ceiling shall be returned in *old_ceiling*.

If the *pthread_mutex_setprioceiling*() function fails, the mutex priority ceiling shall not be changed.

RETURN VALUE

If successful, the *pthread_mutex_getprioceiling*() and *pthread_mutex_setprioceiling*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

EINVAL
The protocol attribute of *mutex* is PTHREAD_PRIO_NONE.

EPERM
The implementation requires appropriate privileges to perform the operation and the caller does not have appropriate privileges.

The *pthread_mutex_setprioceiling*() function shall fail if:

EAGAIN
The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

EDEADLK
The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

EINVAL
The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling, and the implementation adheres to the priority protect protocol in the process of locking the mutex.

ENOTRECOVERABLE
The mutex is a robust mutex and the state protected by the mutex is not recoverable.

EOWNERDEAD
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent (see *pthread_mutex_lock*( )).

The *pthread_mutex_setprioceiling*() function may fail if:

EDEADLK
A deadlock condition was detected.

EINVAL
The priority requested by *prioceiling* is out of range.

EOWNERDEAD
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent (see *pthread_mutex_lock*( )).

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
pthread_mutex_destroy( ), *pthread_mutex_lock*( ), *pthread_mutex_timedlock*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_init — destroy and initialize a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_init(pthread_mutex_t *restrict mutex,
    const pthread_mutexattr_t *restrict attr);
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
```

DESCRIPTION

Refer to *pthread_mutex_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_lock, pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

The mutex object referenced by *mutex* shall be locked by a call to *pthread_mutex_lock*() that returns zero or **[EOWNERDEAD]**. If the mutex is already locked by another thread, the calling thread shall block until the mutex becomes available. This operation shall return with the mutex object referenced by *mutex* in the locked state with the calling thread as its owner. If a thread attempts to relock a mutex that it has already locked, *pthread_mutex_lock*() shall behave as described in the **Relock** column of the following table. If a thread attempts to unlock a mutex that it has not locked or a mutex which is unlocked, *pthread_mutex_unlock*() shall behave as described in the **Unlock When Not Owner** column of the following table.

center box tab(!); cB | cB | cB | cB l | l | l | l. Mutex Type!Robustness!Relock!Unlock When Not Owner _ NORMAL!non-robust!deadlock!undefined behavior _ NORMAL!robust!deadlock!error returned _ ERRORCHECK!either!error returned!error returned _ RECURSIVE!either!recursive!error returned !!(see below) _ DEFAULT!non-robust!undefined!undefined behavior† !!behavior† _ DEFAULT!robust!undefined!error returned !!behavior†

† If the mutex type is PTHREAD_MUTEX_DEFAULT, the behavior of *pthread_mutex_lock*() may correspond to one of the three other standard mutex types as described in the table above. If it does not correspond to one of those three, the behavior is undefined for the cases marked †.

Where the table indicates recursive behavior, the mutex shall maintain the concept of a lock count. When a thread successfully acquires a mutex for the first time, the lock count shall be set to one. Every time a thread relocks this mutex, the lock count shall be incremented by one. Each time the thread unlocks the mutex, the lock count shall be decremented by one. When the lock count reaches zero, the mutex shall become available for other threads to acquire.

The *pthread_mutex_trylock*() function shall be equivalent to *pthread_mutex_lock*(), except that if the mutex object referenced by *mutex* is currently locked (by any thread, including the current thread), the call shall return immediately. If the mutex type is PTHREAD_MUTEX_RECURSIVE and the mutex is currently owned by the calling thread, the mutex lock count shall be incremented by one and the *pthread_mutex_trylock*() function shall immediately return success.

The *pthread_mutex_unlock*() function shall release the mutex object referenced by *mutex*. The manner in which a mutex is released is dependent upon the mutex's type attribute. If there are threads blocked on the mutex object referenced by *mutex* when *pthread_mutex_unlock*() is called, resulting in the mutex becoming available, the scheduling policy shall determine which thread shall acquire the mutex.

(In the case of PTHREAD_MUTEX_RECURSIVE mutexes, the mutex shall become available when the count reaches zero and the calling thread no longer has any locks on this mutex.)

If a signal is delivered to a thread waiting for a mutex, upon return from the signal handler the thread shall resume waiting for the mutex as if it was not interrupted.

If *mutex* is a robust mutex and the process containing the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_lock*() shall return the error value **[EOWNERDEAD]**. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_lock*() may return the error value **[EOWNERDEAD]** even if the process in which the owning thread resides has not terminated. In these cases, the mutex is locked by the thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent*(). If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent*(), after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior of *pthread_mutex_lock*(), *pthread_mutex_trylock*(), and *pthread_mutex_unlock*() is undefined.

RETURN VALUE

If successful, the *pthread_mutex_lock*(), *pthread_mutex_trylock*(), and *pthread_mutex_unlock*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_lock*() and *pthread_mutex_trylock*() functions shall fail if:

EAGAIN
The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

EINVAL
The *mutex* was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex's current priority ceiling.

ENOTRECOVERABLE
The state protected by the mutex is not recoverable.

EOWNERDEAD
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The *pthread_mutex_lock*() function shall fail if:

EDEADLK
The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

The *pthread_mutex_trylock*() function shall fail if:

EBUSY
The *mutex* could not be acquired because it was already locked.

The *pthread_mutex_unlock*() function shall fail if:

EPERM
The mutex type is PTHREAD_MUTEX_ERRORCHECK or PTHREAD_MUTEX_RECURSIVE, or the mutex is a robust mutex, and the current thread does not own the mutex.

The *pthread_mutex_lock*() and *pthread_mutex_trylock*() functions may fail if:

EOWNERDEAD
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

The *pthread_mutex_lock*() function may fail if:

EDEADLK
A deadlock condition was detected.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is **[EOWNERDEAD]**. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

Mutex objects are intended to serve as a low-level primitive from which other thread synchronization functions can be built. As such, the implementation of mutexes should be as efficient as possible, and this has ramifications on the features available at the interface.

The mutex functions and the particular default settings of the mutex attributes have been motivated by the desire to not preclude fast, inlined implementations of mutex locking and unlocking.

Since most attributes only need to be checked when a thread is going to be blocked, the use of attributes does not slow the (common) mutex-locking case.

Likewise, while being able to extract the thread ID of the owner of a mutex might be desirable, it would require storing the current thread ID when each mutex is locked, and this could incur unacceptable levels of overhead. Similar arguments apply to a *mutex_tryunlock* operation.

For further rationale on the extended mutex types, see the Rationale (Informative) volume of POSIX.1-2017, *Threads Extensions*.

If an implementation detects that the value specified by the *mutex* argument does not refer to an initialized mutex object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_consistent( ), *pthread_mutex_destroy*( ), *pthread_mutex_timedlock*( ), *pthread_mutexattr_getrobust*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_setprioceiling — change the priority ceiling of a mutex (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_setprioceiling(pthread_mutex_t *restrict mutex,
    int prioceiling, int *restrict old_ceiling);
```

DESCRIPTION

Refer to *pthread_mutex_getprioceiling*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_timedlock — lock a mutex

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_mutex_timedlock(pthread_mutex_t *restrict mutex,
   const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_mutex_timedlock*() function shall lock the mutex object referenced by *mutex*. If the mutex is already locked, the calling thread shall block until the mutex becomes available as in the *pthread_mutex_lock*() function. If the mutex cannot be locked without waiting for another thread to unlock the mutex, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined in the *<time.h>* header.

Under no circumstance shall the function fail with a timeout if the mutex can be locked immediately. The validity of the *abstime* parameter need not be checked if the mutex can be locked immediately.

As a consequence of the priority inheritance rules (for mutexes initialized with the PRIO_INHERIT protocol), if a timed mutex wait is terminated because its timeout expires, the priority of the owner of the mutex shall be adjusted as necessary to reflect the fact that this thread is no longer among the threads waiting for the mutex.

If *mutex* is a robust mutex and the process containing the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_timedlock*() shall return the error value **[EOWNERDEAD]**. If *mutex* is a robust mutex and the owning thread terminated while holding the mutex lock, a call to *pthread_mutex_timedlock*() may return the error value **[EOWNERDEAD]** even if the process in which the owning thread resides has not terminated. In these cases, the mutex is locked by the thread but the state it protects is marked as inconsistent. The application should ensure that the state is made consistent for reuse and when that is complete call *pthread_mutex_consistent*(). If the application is unable to recover the state, it should unlock the mutex without a prior call to *pthread_mutex_consistent*(), after which the mutex is marked permanently unusable.

If *mutex* does not refer to an initialized mutex object, the behavior is undefined.

RETURN VALUE

If successful, the *pthread_mutex_timedlock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutex_timedlock*() function shall fail if:

EAGAIN
: The mutex could not be acquired because the maximum number of recursive locks for *mutex* has been exceeded.

EDEADLK
: The mutex type is PTHREAD_MUTEX_ERRORCHECK and the current thread already owns the mutex.

EINVAL
The mutex was created with the protocol attribute having the value PTHREAD_PRIO_PROTECT and the calling thread's priority is higher than the mutex' current priority ceiling.

EINVAL
The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

ENOTRECOVERABLE
The state protected by the mutex is not recoverable.

EOWNERDEAD
The mutex is a robust mutex and the process containing the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

ETIMEDOUT
The mutex could not be locked before the specified timeout expired.

The *pthread_mutex_timedlock*() function may fail if:

EDEADLK
A deadlock condition was detected.

EOWNERDEAD
The mutex is a robust mutex and the previous owning thread terminated while holding the mutex lock. The mutex lock shall be acquired by the calling thread and it is up to the new owner to make the state consistent.

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications that have assumed that non-zero return values are errors will need updating for use with robust mutexes, since a valid return for a thread acquiring a mutex which is protecting a currently inconsistent state is **[EOWNERDEAD]**. Applications that do not check the error returns, due to ruling out the possibility of such errors arising, should not use robust mutexes. If an application is supposed to work with normal and robust mutexes, it should check all return values for error conditions and if necessary take appropriate action.

RATIONALE

Refer to *pthread_mutex_lock*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_destroy( ), *pthread_mutex_lock*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutex_trylock, pthread_mutex_unlock — lock and unlock a mutex

SYNOPSIS

```
#include <pthread.h>

int pthread_mutex_trylock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

DESCRIPTION

Refer to *pthread_mutex_lock*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_destroy, pthread_mutexattr_init — destroy and initialize the mutex attributes object

SYNOPSIS

#include <pthread.h>

int pthread_mutexattr_destroy(pthread_mutexattr_t **attr*);
int pthread_mutexattr_init(pthread_mutexattr_t **attr*);

DESCRIPTION

The *pthread_mutexattr_destroy*() function shall destroy a mutex attributes object; the object becomes, in effect, uninitialized. An implementation may cause *pthread_mutexattr_destroy*() to set the object referenced by *attr* to an invalid value.

A destroyed *attr* attributes object can be reinitialized using *pthread_mutexattr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined.

The *pthread_mutexattr_init*() function shall initialize a mutex attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_mutexattr_init*() is called specifying an already initialized *attr* attributes object.

After a mutex attributes object has been used to initialize one or more mutexes, any function affecting the attributes object (including destruction) shall not affect any previously initialized mutexes.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_destroy*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_destroy*() and *pthread_mutexattr_init*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the mutex attributes object.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_destroy*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

See *pthread_attr_destroy*( ) for a general explanation of attributes. Attributes objects allow implementations to experiment with useful extensions and permit extension of this volume of POSIX.1-2017 without changing the existing functions. Thus, they provide for future extensibility of this volume of POSIX.1-2017 and reduce the temptation to standardize prematurely on semantics that are not yet widely implemented or understood.

Examples of possible additional mutex attributes that have been discussed are *spin_only*, *limited_spin*, *no_spin*, *recursive*, and *metered*. (To explain what the latter attributes might mean: recursive mutexes would allow for multiple re-locking by the current owner; metered mutexes would transparently keep records of queue length, wait time, and so on.) Since there is not yet wide agreement on the usefulness of these resulting from shared implementation and usage experience, they are not yet specified in this volume of POSIX.1-2017. Mutex attributes objects, however, make it possible to test out these concepts for possible standardization at a later time.

Mutex Attributes and Performance

Care has been taken to ensure that the default values of the mutex attributes have been defined such that mutexes initialized with the defaults have simple enough semantics so that the locking and unlocking can be done with the equivalent of a test-and-set instruction (plus possibly a few other basic instructions).

There is at least one implementation method that can be used to reduce the cost of testing at lock-time if a mutex has non-default attributes. One such method that an implementation can employ (and this can be made fully transparent to fully conforming POSIX applications) is to secretly pre-lock any mutexes that are initialized to non-default attributes. Any later attempt to lock such a mutex causes the implementation to branch to the ``slow path'' as if the mutex were unavailable; then, on the slow path, the implementation can do the ``real work'' to lock a non-default mutex. The underlying unlock operation is more complicated since the implementation never really wants to release the pre-lock on this kind of mutex. This illustrates that, depending on the hardware, there may be certain optimizations that can be used so that whatever mutex attributes are considered ``most frequently used'' can be processed most efficiently.

Process Shared Memory and Synchronization

The existence of memory mapping functions in this volume of POSIX.1-2017 leads to the possibility that an application may allocate the synchronization objects from this section in memory that is accessed by multiple processes (and therefore, by threads of multiple processes).

In order to permit such usage, while at the same time keeping the usual case (that is, usage within a single process) efficient, a *process-shared* option has been defined.

If an implementation supports the _POSIX_THREAD_PROCESS_SHARED option, then the *process-shared* attribute can be used to indicate that mutexes or condition variables may be accessed by threads of multiple processes.

The default setting of PTHREAD_PROCESS_PRIVATE has been chosen for the *process-shared* attribute so that the most efficient forms of these synchronization objects are created by default.

Synchronization variables that are initialized with the PTHREAD_PROCESS_PRIVATE *process-shared* attribute may only be operated on by threads in the process that initialized them. Synchronization variables that are initialized with the PTHREAD_PROCESS_SHARED *process-shared* attribute may be operated on by any thread in any process that has access to it. In particular, these processes may exist beyond the lifetime of the initializing process. For example, the following code implements a simple counting semaphore in a mapped file that may be used by many processes.

```
/* sem.h */
struct semaphore {
    pthread_mutex_t lock;
    pthread_cond_t nonzero;
    unsigned count;
};
typedef struct semaphore semaphore_t;

semaphore_t *semaphore_create(char *semaphore_name);
semaphore_t *semaphore_open(char *semaphore_name);
void semaphore_post(semaphore_t *semap);
void semaphore_wait(semaphore_t *semap);
void semaphore_close(semaphore_t *semap);

/* sem.c */
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <pthread.h>
#include "sem.h"

semaphore_t *
semaphore_create(char *semaphore_name)
{
int fd;
    semaphore_t *semap;
    pthread_mutexattr_t psharedm;
    pthread_condattr_t psharedc;

    fd = open(semaphore_name, O_RDWR | O_CREAT | O_EXCL, 0666);
    if (fd < 0)
        return (NULL);
    (void) ftruncate(fd, sizeof(semaphore_t));
    (void) pthread_mutexattr_init(&psharedm);
    (void) pthread_mutexattr_setpshared(&psharedm,
        PTHREAD_PROCESS_SHARED);
    (void) pthread_condattr_init(&psharedc);
    (void) pthread_condattr_setpshared(&psharedc,
        PTHREAD_PROCESS_SHARED);
    semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
            PROT_READ | PROT_WRITE, MAP_SHARED,
            fd, 0);
    close (fd);
    (void) pthread_mutex_init(&semap->lock, &psharedm);
    (void) pthread_cond_init(&semap->nonzero, &psharedc);
    semap->count = 0;
    return (semap);
}

semaphore_t *
semaphore_open(char *semaphore_name)
{
    int fd;
    semaphore_t *semap;

    fd = open(semaphore_name, O_RDWR, 0666);
    if (fd < 0)
        return (NULL);
    semap = (semaphore_t *) mmap(NULL, sizeof(semaphore_t),
            PROT_READ | PROT_WRITE, MAP_SHARED,
            fd, 0);
    close (fd);
    return (semap);
}

void
semaphore_post(semaphore_t *semap)
{
    pthread_mutex_lock(&semap->lock);
    if (semap->count == 0)
        pthread_cond_signal(&semapx->nonzero);
    semap->count++;
    pthread_mutex_unlock(&semap->lock);
}

void
semaphore_wait(semaphore_t *semap)
{
    pthread_mutex_lock(&semap->lock);
    while (semap->count == 0)
        pthread_cond_wait(&semap->nonzero, &semap->lock);
    semap->count--;
    pthread_mutex_unlock(&semap->lock);
}

void
semaphore_close(semaphore_t *semap)
{
    munmap((void *) semap, sizeof(semaphore_t));
}
```

The following code is for three separate processes that create, post, and wait on a semaphore in the file **/tmp/semaphore**. Once the file is created, the post and wait programs increment and decrement the counting semaphore (waiting and waking as required) even though they did not initialize the semaphore.

```
/* create.c */
#include "pthread.h"
#include "sem.h"

int
main()
{
    semaphore_t *semap;

    semap = semaphore_create("/tmp/semaphore");
    if (semap == NULL)
        exit(1);
    semaphore_close(semap);
    return (0);
}

/* post */
#include "pthread.h"
#include "sem.h"

int
main()
{
    semaphore_t *semap;

    semap = semaphore_open("/tmp/semaphore");
    if (semap == NULL)
        exit(1);
    semaphore_post(semap);
    semaphore_close(semap);
    return (0);
}

/* wait */
#include "pthread.h"
```

```
#include "sem.h"

int
main()
{
    semaphore_t *semap;

    semap = semaphore_open("/tmp/semaphore");
    if (semap == NULL)
        exit(1);
    semaphore_wait(semap);
    semaphore_close(semap);
    return (0);
}
```

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_getprioceiling, pthread_mutexattr_setprioceiling — get and set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getprioceiling(const pthread_mutexattr_t
    *restrict attr, int *restrict prioceiling);
int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

DESCRIPTION

The *pthread_mutexattr_getprioceiling*() and *pthread_mutexattr_setprioceiling*() functions, respectively, shall get and set the priority ceiling attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*().

The *prioceiling* attribute contains the priority ceiling of initialized mutexes. The values of *prioceiling* are within the maximum range of priorities defined by SCHED_FIFO.

The *prioceiling* attribute defines the priority ceiling of initialized mutexes, which is the minimum priority level at which the critical section guarded by the mutex is executed. In order to avoid priority inversion, the priority ceiling of the mutex shall be set to a priority higher than or equal to the highest priority of all the threads that may lock that mutex. The values of *prioceiling* are within the maximum range of priorities defined under the SCHED_FIFO scheduling policy.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getprioceiling*() or *pthread_mutexattr_setprioceiling*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprioceiling*() and *pthread_mutexattr_setprioceiling*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value specified by *prioceiling* is invalid.

EPERM
The caller does not have the privilege to perform the operation.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getprioceiling*() or *pthread_mutexattr_setprioceiling*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_getprotocol, pthread_mutexattr_setprotocol — get and set the protocol attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getprotocol(const pthread_mutexattr_t
    *restrict attr, int *restrict protocol);
int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

DESCRIPTION

The *pthread_mutexattr_getprotocol*() and *pthread_mutexattr_setprotocol*() functions, respectively, shall get and set the protocol attribute of a mutex attributes object pointed to by *attr* which was previously created by the function *pthread_mutexattr_init*().

The *protocol* attribute defines the protocol to be followed in utilizing mutexes. The value of *protocol* may be one of:

PTHREAD_PRIO_INHERIT
PTHREAD_PRIO_NONE
PTHREAD_PRIO_PROTECT

which are defined in the *<pthread.h>* header. The default value of the attribute shall be PTHREAD_PRIO_NONE.

When a thread owns a mutex with the PTHREAD_PRIO_NONE *protocol* attribute, its priority and scheduling shall not be affected by its mutex ownership.

When a thread is blocking higher priority threads because of owning one or more robust mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or the priority of the highest priority thread waiting on any of the robust mutexes owned by this thread and initialized with this protocol.

When a thread is blocking higher priority threads because of owning one or more non-robust mutexes with the PTHREAD_PRIO_INHERIT *protocol* attribute, it shall execute at the higher of its priority or the priority of the highest priority thread waiting on any of the non-robust mutexes owned by this thread and initialized with this protocol.

When a thread owns one or more robust mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all the robust mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these robust mutexes or not.

When a thread owns one or more non-robust mutexes initialized with the PTHREAD_PRIO_PROTECT protocol, it shall execute at the higher of its priority or the highest of the priority ceilings of all the non-robust mutexes owned by this thread and initialized with this attribute, regardless of whether other threads are blocked on any of these non-robust mutexes or not.

While a thread is holding a mutex which has been initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed, such as by a call to *sched_setparam*(). Likewise, when a thread unlocks a mutex that has been initialized with the PTHREAD_PRIO_INHERIT or PTHREAD_PRIO_PROTECT protocol attributes, it shall not be subject to being moved to the tail of the scheduling queue at its priority in the event that its original priority is changed.

If a thread simultaneously owns several mutexes initialized with different protocols, it shall execute at the highest of the priorities that it would have obtained by each of these protocols.

When a thread makes a call to *pthread_mutex_lock*(), the mutex was initialized with the protocol attribute having the value PTHREAD_PRIO_INHERIT, when the calling thread is blocked because the mutex is owned by another thread, that owner thread shall inherit the priority level of the calling thread as long as it continues to own the mutex. The implementation shall update its execution priority to the maximum of its assigned priority and all its inherited priorities. Furthermore, if this owner thread itself becomes blocked on another mutex with the *protocol* attribute having the value PTHREAD_PRIO_INHERIT, the same priority inheritance effect shall be propagated to this other owner thread, in a recursive manner.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getprotocol*() or *pthread_mutexattr_setprotocol*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getprotocol*() and *pthread_mutexattr_setprotocol*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setprotocol*() function shall fail if:

ENOTSUP
The value specified by *protocol* is an unsupported value.

The *pthread_mutexattr_getprotocol*() and *pthread_mutexattr_setprotocol*() functions may fail if:

EINVAL
The value specified by *protocol* is invalid.

EPERM
The caller does not have the privilege to perform the operation.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getprotocol*() or *pthread_mutexattr_setprotocol*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_getpshared, pthread_mutexattr_setpshared — get and set the process-shared attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getpshared(const pthread_mutexattr_t
    *restrict attr, int *restrict pshared);
int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

The *pthread_mutexattr_getpshared*() function shall obtain the value of the *process-shared* attribute from the attributes object referenced by *attr*.

The *pthread_mutexattr_setpshared*() function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute is set to PTHREAD_PROCESS_SHARED to permit a mutex to be operated upon by any thread that has access to the memory where the mutex is allocated, even if the mutex is allocated in memory that is shared by multiple processes. See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements. The default value of the attribute shall be PTHREAD_PROCESS_PRIVATE.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getpshared*() or *pthread_mutexattr_setpshared*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, *pthread_mutexattr_setpshared*() shall return zero; otherwise, an error number shall be returned to indicate the error.

Upon successful completion, *pthread_mutexattr_getpshared*() shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setpshared*() function may fail if:

EINVAL

The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getpshared*() or *pthread_mutexattr_setpshared*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_destroy( ), *pthread_create*( ), *pthread_mutex_destroy*( ), *pthread_mutexattr_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_getrobust, pthread_mutexattr_setrobust — get and set the mutex robust attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_getrobust(const pthread_mutexattr_t *restrict
    attr, int *restrict robust);
int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
    int robust);
```

DESCRIPTION

The *pthread_mutexattr_getrobust*() and *pthread_mutexattr_setrobust*() functions, respectively, shall get and set the mutex *robust* attribute. This attribute is set in the *robust* parameter. Valid values for *robust* include:

PTHREAD_MUTEX_STALLED
: No special actions are taken if the owner of the mutex is terminated while holding the mutex lock. This can lead to deadlocks if no other thread can unlock the mutex.
This is the default value.

PTHREAD_MUTEX_ROBUST
: If the process containing the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that acquires the mutex shall be notified about the termination by the return value **[EOWNERDEAD]** from the locking function. If the owning thread of a robust mutex terminates while holding the mutex lock, the next thread that attempts to acquire the mutex may be notified about the termination by the return value **[EOWNERDEAD]**. The notified thread can then attempt to make the state protected by the mutex consistent again, and if successful can mark the mutex state as consistent by calling *pthread_mutex_consistent*(). After a subsequent successful call to *pthread_mutex_unlock*(), the mutex lock shall be released and can be used normally by other threads. If the mutex is unlocked without a call to *pthread_mutex_consistent*(), it shall be in a permanently unusable state and all attempts to lock the mutex shall fail with the error **[ENOTRECOVERABLE]**. The only permissible operation on such a mutex is *pthread_mutex_destroy*().

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_getrobust*() or *pthread_mutexattr_setrobust*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_getrobust*() function shall return zero and store the value of the *robust* attribute of *attr* into the object referenced by the *robust* parameter. Otherwise, an error value shall be returned to indicate the error. If successful, the *pthread_mutexattr_setrobust*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_setrobust*() function shall fail if:

EINVAL
: The value of *robust* is invalid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The actions required to make the state protected by the mutex consistent again are solely dependent on the application. If it is not possible to make the state of a mutex consistent, robust mutexes can be used to notify this situation by calling *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*().

If the state is declared inconsistent by calling *pthread_mutex_unlock*() without a prior call to *pthread_mutex_consistent*(), a possible approach could be to destroy the mutex and then reinitialize it. However, it should be noted that this is possible only in certain situations where the state protected by the mutex has to be reinitialized and coordination achieved with other threads blocked on the mutex, because otherwise a call to a locking function with a reference to a mutex object invalidated by a call to *pthread_mutex_destroy*() results in undefined behavior.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_getrobust*() or *pthread_mutexattr_setrobust*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_mutex_consistent( ), *pthread_mutex_destroy*( ), *pthread_mutex_lock*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_gettype, pthread_mutexattr_settype — get and set the mutex type attribute

SYNOPSIS

#include <pthread.h>

int pthread_mutexattr_gettype(const pthread_mutexattr_t *restrict *attr*,
    int *restrict *type*);
int pthread_mutexattr_settype(pthread_mutexattr_t **attr*, int *type*);

DESCRIPTION

The *pthread_mutexattr_gettype*() and *pthread_mutexattr_settype*() functions, respectively, shall get and set the mutex *type* attribute. This attribute is set in the *type* parameter to these functions. The default value of the *type* attribute is PTHREAD_MUTEX_DEFAULT.

The type of mutex is contained in the *type* attribute of the mutex attributes. Valid mutex types include:

PTHREAD_MUTEX_NORMAL PTHREAD_MUTEX_ERRORCHECK PTHREAD_MUTEX_RECURSIVE PTHREAD_MUTEX_DEFAULT

The mutex type affects the behavior of calls which lock and unlock the mutex. See *pthread_mutex_lock*( ) for details. An implementation may map PTHREAD_MUTEX_DEFAULT to one of the other mutex types.

The behavior is undefined if the value specified by the *attr* argument to *pthread_mutexattr_gettype*() or *pthread_mutexattr_settype*() does not refer to an initialized mutex attributes object.

RETURN VALUE

Upon successful completion, the *pthread_mutexattr_gettype*() function shall return zero and store the value of the *type* attribute of *attr* into the object referenced by the *type* parameter. Otherwise, an error shall be returned to indicate the error.

If successful, the *pthread_mutexattr_settype*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_mutexattr_settype*() function shall fail if:

EINVAL
    The value *type* is invalid.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

It is advised that an application should not use a PTHREAD_MUTEX_RECURSIVE mutex with condition variables because the implicit unlock performed for a *pthread_cond_timedwait*() or *pthread_cond_wait*() may not actually release the mutex (if it had been locked multiple times). If this happens, no other thread can satisfy the condition of the predicate.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_mutexattr_gettype*() or *pthread_mutexattr_settype*() does not refer to an initialized mutex attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_cond_timedwait( ), *pthread_mutex_lock*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_init — initialize the mutex attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_init(pthread_mutexattr_t *attr);
```

DESCRIPTION

Refer to *pthread_mutexattr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_setprioceiling — set the prioceiling attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_setprioceiling(pthread_mutexattr_t *attr,
    int prioceiling);
```

DESCRIPTION

Refer to *pthread_mutexattr_getprioceiling*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_setprotocol — set the protocol attribute of the mutex attributes object (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_setprotocol(pthread_mutexattr_t *attr,
    int protocol);
```

DESCRIPTION

Refer to *pthread_mutexattr_getprotocol*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_setpshared — set the process-shared attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_setpshared(pthread_mutexattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_mutexattr_getpshared*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_setrobust — get and set the mutex robust attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_setrobust(pthread_mutexattr_t *attr,
   int robust);
```

DESCRIPTION

Refer to *pthread_mutexattr_getrobust*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_mutexattr_settype — set the mutex type attribute

SYNOPSIS

```
#include <pthread.h>

int pthread_mutexattr_settype(pthread_mutexattr_t *attr, int type);
```

DESCRIPTION

Refer to *pthread_mutexattr_gettype*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_once — dynamic package initialization

SYNOPSIS

```
#include <pthread.h>

int pthread_once(pthread_once_t *once_control,
    void (*init_routine)(void));
pthread_once_t once_control = PTHREAD_ONCE_INIT;
```

DESCRIPTION

The first call to *pthread_once*() by any thread in a process, with a given *once_control*, shall call the *init_routine* with no arguments. Subsequent calls of *pthread_once*() with the same *once_control* shall not call the *init_routine*. On return from *pthread_once*(), *init_routine* shall have completed. The *once_control* parameter shall determine whether the associated initialization routine has been called.

The *pthread_once*() function is not a cancellation point. However, if *init_routine* is a cancellation point and is canceled, the effect on *once_control* shall be as if *pthread_once*() was never called.

If the call to *init_routine* is terminated by a call to *longjmp*(), *_longjmp*(), or *siglongjmp*(), the behavior is undefined.

The constant PTHREAD_ONCE_INIT is defined in the *<pthread.h>* header.

The behavior of *pthread_once*() is undefined if *once_control* has automatic storage duration or is not initialized by PTHREAD_ONCE_INIT.

RETURN VALUE

Upon successful completion, *pthread_once*() shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_once*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If *init_routine* recursively calls *pthread_once*() with the same *once_control*, the recursive call will not call the specified *init_routine*, and thus the specified *init_routine* will not complete, and thus the recursive call to *pthread_once*() will not return. Use of *longjmp*(), *_longjmp*(), or *siglongjmp*() within an *init_routine* to jump to a point outside of *init_routine* prevents *init_routine* from returning.

RATIONALE

Some C libraries are designed for dynamic initialization. That is, the global initialization for the library is performed when the first procedure in the library is called. In a single-threaded program, this is normally implemented using a static variable whose value is checked on entry to a routine, as follows:

```
static int random_is_initialized = 0;
extern void initialize_random(void);

int random_function()
{
    if (random_is_initialized == 0) {
        initialize_random();
        random_is_initialized = 1;
    }
    ... /* Operations performed after initialization. */
}
```

To keep the same structure in a multi-threaded program, a new primitive is needed. Otherwise, library initialization has to be accomplished by an explicit call to a library-exported initialization function prior to any use of the library.

For dynamic library initialization in a multi-threaded process, if an initialization flag is used the flag needs to be protected against modification by multiple threads simultaneously calling into the library. This can be done by using a mutex (initialized by assigning PTHREAD_MUTEX_INITIALIZER). However, the better solution is to use *pthread_once*() which is designed for exactly this purpose, as follows:

```
#include <pthread.h>
static pthread_once_t random_is_initialized = PTHREAD_ONCE_INIT;
extern void initialize_random(void);

int random_function()
{
    (void) pthread_once(&random_is_initialized, initialize_random);
    ... /* Operations performed after initialization. */
}
```

If an implementation detects that the value specified by the *once_control* argument to *pthread_once*() does not refer to a **pthread_once_t** object initialized by PTHREAD_ONCE_INIT, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_destroy, pthread_rwlock_init — destroy and initialize a read-write lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_destroy(pthread_rwlock_t *rwlock);
int pthread_rwlock_init(pthread_rwlock_t *restrict rwlock,
    const pthread_rwlockattr_t *restrict attr);
pthread_rwlock_t rwlock = PTHREAD_RWLOCK_INITIALIZER;
```

DESCRIPTION

The *pthread_rwlock_destroy*() function shall destroy the read-write lock object referenced by *rwlock* and release any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is reinitialized by another call to *pthread_rwlock_init*(). An implementation may cause *pthread_rwlock_destroy*() to set the object referenced by *rwlock* to an invalid value. Results are undefined if *pthread_rwlock_destroy*() is called when any thread holds *rwlock*. Attempting to destroy an uninitialized read-write lock results in undefined behavior.

The *pthread_rwlock_init*() function shall allocate any resources required to use the read-write lock referenced by *rwlock* and initializes the lock to an unlocked state with attributes referenced by *attr*. If *attr* is NULL, the default read-write lock attributes shall be used; the effect is the same as passing the address of a default read-write lock attributes object. Once initialized, the lock can be used any number of times without being reinitialized. Results are undefined if *pthread_rwlock_init*() is called specifying an already initialized read-write lock. Results are undefined if a read-write lock is used without first being initialized.

If the *pthread_rwlock_init*() function fails, *rwlock* shall not be initialized and the contents of *rwlock* are undefined.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

In cases where default read-write lock attributes are appropriate, the macro PTHREAD_RWLOCK_INITIALIZER can be used to initialize read-write locks. The effect shall be equivalent to dynamic initialization by a call to *pthread_rwlock_init*() with the *attr* parameter specified as NULL, except that no error checks are performed.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlock_init*() does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the *pthread_rwlock_destroy*() and *pthread_rwlock_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_init*() function shall fail if:

EAGAIN
The system lacked the necessary resources (other than memory) to initialize another read-write lock.

ENOMEM
Insufficient memory exists to initialize the read-write lock.

EPERM
The caller does not have the privilege to perform the operation.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these and related read-write lock functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *attr* argument to *pthread_rwlock_init*() does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_destroy*() or *pthread_rwlock_init*() refers to a locked read-write lock object, or detects that the value specified by the *rwlock* argument to *pthread_rwlock_init*() refers to an already initialized read-write lock object, it is recommended that the function should fail and report an **[EBUSY]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_rdlock( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_rdlock, pthread_rwlock_tryrdlock — lock a read-write lock object for reading

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_rdlock(pthread_rwlock_t *rwlock);
int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

The *pthread_rwlock_rdlock*() function shall apply a read lock to the read-write lock referenced by *rwlock*. The calling thread acquires the read lock if a writer does not hold the lock and there are no writers blocked on the lock.

If the Thread Execution Scheduling option is supported, and the threads involved in the lock are executing with the scheduling policies SCHED_FIFO or SCHED_RR, the calling thread shall not acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock; otherwise, the calling thread shall acquire the lock.

If the Thread Execution Scheduling option is supported, and the threads involved in the lock are executing with the SCHED_SPORADIC scheduling policy, the calling thread shall not acquire the lock if a writer holds the lock or if writers of higher or equal priority are blocked on the lock; otherwise, the calling thread shall acquire the lock.

If the Thread Execution Scheduling option is not supported, it is implementation-defined whether the calling thread acquires the lock when a writer does not hold the lock and there are writers blocked on the lock. If a writer holds the lock, the calling thread shall not acquire the read lock. If the read lock is not acquired, the calling thread shall block until it can acquire the lock. The calling thread may deadlock if at the time the call is made it holds a write lock.

A thread may hold multiple concurrent read locks on *rwlock* (that is, successfully call the *pthread_rwlock_rdlock*() function *n* times). If so, the application shall ensure that the thread performs matching unlocks (that is, it calls the *pthread_rwlock_unlock*() function *n* times).

The maximum number of simultaneous read locks that an implementation guarantees can be applied to a read-write lock shall be implementation-defined. The *pthread_rwlock_rdlock*() function may fail if this maximum would be exceeded.

The *pthread_rwlock_tryrdlock*() function shall apply a read lock as in the *pthread_rwlock_rdlock*() function, with the exception that the function shall fail if the equivalent *pthread_rwlock_rdlock*() call would have blocked the calling thread. In no case shall the *pthread_rwlock_tryrdlock*() function ever block; it always either acquires the lock or fails and returns immediately.

Results are undefined if any of these functions are called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for reading, upon return from the signal handler the thread resumes waiting for the read-write lock for reading as if it was not interrupted.

RETURN VALUE

If successful, the *pthread_rwlock_rdlock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

The *pthread_rwlock_tryrdlock*() function shall return zero if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_tryrdlock*() function shall fail if:

EBUSY
The read-write lock could not be acquired for reading because a writer holds the lock or a writer with the appropriate priority was blocked on it.

The *pthread_rwlock_rdlock*() and *pthread_rwlock_tryrdlock*() functions may fail if:

EAGAIN
The read lock could not be acquired because the maximum number of read locks for *rwlock* has been exceeded.

The *pthread_rwlock_rdlock*() function may fail if:

EDEADLK
A deadlock condition was detected or the current thread already owns the read-write lock for writing.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_rdlock*() or *pthread_rwlock_tryrdlock*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_timedrdlock — lock a read-write lock for reading

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedrdlock(pthread_rwlock_t *restrict rwlock,
   const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_timedrdlock*() function shall apply a read lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_rdlock*() function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the *<time.h>* header. Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_timedrdlock*(), upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds a write lock on *rwlock*. The results are undefined if this function is called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_timedrdlock*() function shall return zero if the lock for reading on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_timedrdlock*() function shall fail if:

ETIMEDOUT
The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_timedrdlock*() function may fail if:

EAGAIN
The read lock could not be acquired because the maximum number of read locks for lock would be exceeded.

EDEADLK
A deadlock condition was detected or the calling thread already holds a write lock on *rwlock*.

EINVAL
The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million.

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_timedrdlock*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_timedwrlock — lock a read-write lock for writing

SYNOPSIS

```
#include <pthread.h>
#include <time.h>

int pthread_rwlock_timedwrlock(pthread_rwlock_t *restrict rwlock,
    const struct timespec *restrict abstime);
```

DESCRIPTION

The *pthread_rwlock_timedwrlock*() function shall apply a write lock to the read-write lock referenced by *rwlock* as in the *pthread_rwlock_wrlock*() function. However, if the lock cannot be acquired without waiting for other threads to unlock the lock, this wait shall be terminated when the specified timeout expires. The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the CLOCK_REALTIME clock. The **timespec** data type is defined in the *<time.h>* header. Under no circumstances shall the function fail with a timeout if the lock can be acquired immediately. The validity of the *abstime* parameter need not be checked if the lock can be immediately acquired.

If a signal that causes a signal handler to be executed is delivered to a thread blocked on a read-write lock via a call to *pthread_rwlock_timedwrlock*(), upon return from the signal handler the thread shall resume waiting for the lock as if it was not interrupted.

The calling thread may deadlock if at the time the call is made it holds the read-write lock. The results are undefined if this function is called with an uninitialized read-write lock.

RETURN VALUE

The *pthread_rwlock_timedwrlock*() function shall return zero if the lock for writing on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_timedwrlock*() function shall fail if:

ETIMEDOUT
The lock could not be acquired before the specified timeout expired.

The *pthread_rwlock_timedwrlock*() function may fail if:

EDEADLK
A deadlock condition was detected or the calling thread already holds the *rwlock*.

EINVAL
The *abstime* nanosecond value is less than zero or greater than or equal to 1 000 million.

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_timedwrlock*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_trywrlock*( ), *pthread_rwlock_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_tryrdlock — lock a read-write lock object for reading

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_tryrdlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

Refer to *pthread_rwlock_rdlock*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_trywrlock, pthread_rwlock_wrlock — lock a read-write lock object for writing

SYNOPSIS

#include <pthread.h>

int pthread_rwlock_trywrlock(pthread_rwlock_t **rwlock*);
int pthread_rwlock_wrlock(pthread_rwlock_t **rwlock*);

DESCRIPTION

The *pthread_rwlock_trywrlock*() function shall apply a write lock like the *pthread_rwlock_wrlock*() function, with the exception that the function shall fail if any thread currently holds *rwlock* (for reading or writing).

The *pthread_rwlock_wrlock*() function shall apply a write lock to the read-write lock referenced by *rwlock*. The calling thread shall acquire the write lock if no thread (reader or writer) holds the read-write lock *rwlock*. Otherwise, if another thread holds the read-write lock *rwlock*, the calling thread shall block until it can acquire the lock. If a deadlock condition occurs or the calling thread already owns the read-write lock for writing or reading, the call shall either deadlock or return **[EDEADLK]**.

Results are undefined if any of these functions are called with an uninitialized read-write lock.

If a signal is delivered to a thread waiting for a read-write lock for writing, upon return from the signal handler the thread resumes waiting for the read-write lock for writing as if it was not interrupted.

RETURN VALUE

The *pthread_rwlock_trywrlock*() function shall return zero if the lock for writing on the read-write lock object referenced by *rwlock* is acquired. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlock_wrlock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_trywrlock*() function shall fail if:

EBUSY
The read-write lock could not be acquired for writing because it was already locked for reading or writing.

The *pthread_rwlock_wrlock*() function may fail if:

EDEADLK
A deadlock condition was detected or the current thread already owns the read-write lock for writing or reading.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_trywrlock*() or *pthread_rwlock_wrlock*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_unlock — unlock a read-write lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlock_unlock(pthread_rwlock_t *rwlock);
```

DESCRIPTION

The *pthread_rwlock_unlock*() function shall release a lock held on the read-write lock object referenced by *rwlock*. Results are undefined if the read-write lock *rwlock* is not held by the calling thread.

If this function is called to release a read lock from the read-write lock object and there are other read locks currently held on this read-write lock object, the read-write lock object remains in the read locked state. If this function releases the last read lock for this read-write lock object, the read-write lock object shall be put in the unlocked state with no owners.

If this function is called to release a write lock for this read-write lock object, the read-write lock object shall be put in the unlocked state.

If there are threads blocked on the lock when it becomes available, the scheduling policy shall determine which thread(s) shall acquire the lock. If the Thread Execution Scheduling option is supported, when threads executing with the scheduling policies SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC are waiting on the lock, they shall acquire the lock in priority order when the lock becomes available. For equal priority threads, write locks shall take precedence over read locks. If the Thread Execution Scheduling option is not supported, it is implementation-defined whether write locks take precedence over read locks.

Results are undefined if this function is called with an uninitialized read-write lock.

RETURN VALUE

If successful, the *pthread_rwlock_unlock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlock_unlock*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_unlock*() does not refer to an initialized read-write lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *rwlock* argument to *pthread_rwlock_unlock*() refers to a read-write lock object for which the current thread does not hold a lock, it is recommended that the function should fail and report an **[EPERM]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlock_rdlock*( ), *pthread_rwlock_timedrdlock*( ), *pthread_rwlock_timedwrlock*( ), *pthread_rwlock_trywrlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlock_wrlock — lock a read-write lock object for writing

SYNOPSIS

#include <pthread.h>

int pthread_rwlock_wrlock(pthread_rwlock_t **rwlock*);

DESCRIPTION

Refer to *pthread_rwlock_trywrlock*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlockattr_destroy, pthread_rwlockattr_init — destroy and initialize the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_destroy(pthread_rwlockattr_t *attr);
int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

DESCRIPTION

The *pthread_rwlockattr_destroy*() function shall destroy a read-write lock attributes object. A destroyed *attr* attributes object can be reinitialized using *pthread_rwlockattr_init*(); the results of otherwise referencing the object after it has been destroyed are undefined. An implementation may cause *pthread_rwlockattr_destroy*() to set the object referenced by *attr* to an invalid value.

The *pthread_rwlockattr_init*() function shall initialize a read-write lock attributes object *attr* with the default value for all of the attributes defined by the implementation.

Results are undefined if *pthread_rwlockattr_init*() is called specifying an already initialized *attr* attributes object.

After a read-write lock attributes object has been used to initialize one or more read-write locks, any function affecting the attributes object (including destruction) shall not affect any previously initialized read-write locks.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlockattr_destroy*() does not refer to an initialized read-write lock attributes object.

RETURN VALUE

If successful, the *pthread_rwlockattr_destroy*() and *pthread_rwlockattr_init*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_init*() function shall fail if:

ENOMEM
Insufficient memory exists to initialize the read-write lock attributes object.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *attr* argument to *pthread_rwlockattr_destroy*() does not refer to an initialized read-write lock attributes object, it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlockattr_getpshared*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlockattr_getpshared, pthread_rwlockattr_setpshared — get and set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_getpshared(const pthread_rwlockattr_t
   *restrict attr, int *restrict pshared);
int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
   int pshared);
```

DESCRIPTION

The *pthread_rwlockattr_getpshared*() function shall obtain the value of the *process-shared* attribute from the initialized attributes object referenced by *attr*. The *pthread_rwlockattr_setpshared*() function shall set the *process-shared* attribute in an initialized attributes object referenced by *attr*.

The *process-shared* attribute shall be set to PTHREAD_PROCESS_SHARED to permit a read-write lock to be operated upon by any thread that has access to the memory where the read-write lock is allocated, even if the read-write lock is allocated in memory that is shared by multiple processes. See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements. The default value of the *process-shared* attribute shall be PTHREAD_PROCESS_PRIVATE.

Additional attributes, their default values, and the names of the associated functions to get and set those attribute values are implementation-defined.

The behavior is undefined if the value specified by the *attr* argument to *pthread_rwlockattr_getpshared*() or *pthread_rwlockattr_setpshared*() does not refer to an initialized read-write lock attributes object.

RETURN VALUE

Upon successful completion, the *pthread_rwlockattr_getpshared*() function shall return zero and store the value of the *process-shared* attribute of *attr* into the object referenced by the *pshared* parameter. Otherwise, an error number shall be returned to indicate the error.

If successful, the *pthread_rwlockattr_setpshared*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_rwlockattr_setpshared*() function may fail if:

EINVAL
The new value specified for the attribute is outside the range of legal values for that attribute.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_rwlock_destroy( ), *pthread_rwlockattr_destroy*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlockattr_init — initialize the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_init(pthread_rwlockattr_t *attr);
```

DESCRIPTION

Refer to *pthread_rwlockattr_destroy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_rwlockattr_setpshared — set the process-shared attribute of the read-write lock attributes object

SYNOPSIS

```
#include <pthread.h>

int pthread_rwlockattr_setpshared(pthread_rwlockattr_t *attr,
    int pshared);
```

DESCRIPTION

Refer to *pthread_rwlockattr_getpshared*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_self — get the calling thread ID

SYNOPSIS

```
#include <pthread.h>

pthread_t pthread_self(void);
```

DESCRIPTION

The *pthread_self*() function shall return the thread ID of the calling thread.

RETURN VALUE

The *pthread_self*() function shall always be successful and no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_self*() function provides a capability similar to the *getpid*() function for processes and the rationale is the same: the creation call does not provide the thread ID to the created thread.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_create( ), *pthread_equal*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_setcancelstate, pthread_setcanceltype, pthread_testcancel — set cancelability state

SYNOPSIS

```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
void pthread_testcancel(void);
```

DESCRIPTION

The *pthread_setcancelstate*() function shall atomically both set the calling thread's cancelability state to the indicated *state* and return the previous cancelability state at the location referenced by *oldstate*. Legal values for *state* are PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*() function shall atomically both set the calling thread's cancelability type to the indicated *type* and return the previous cancelability type at the location referenced by *oldtype*. Legal values for *type* are PTHREAD_CANCEL_DEFERRED and PTHREAD_CANCEL_ASYNCHRONOUS.

The cancelability state and type of any newly created threads, including the thread in which *main*() was first invoked, shall be PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED respectively.

The *pthread_testcancel*() function shall create a cancellation point in the calling thread. The *pthread_testcancel*() function shall have no effect if cancelability is disabled.

RETURN VALUE

If successful, the *pthread_setcancelstate*() and *pthread_setcanceltype*() functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setcancelstate*() function may fail if:

EINVAL
The specified state is not PTHREAD_CANCEL_ENABLE or PTHREAD_CANCEL_DISABLE.

The *pthread_setcanceltype*() function may fail if:

EINVAL
The specified type is not PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

In order to write a signal handler for an asynchronous signal which can run safely in a cancellable thread, *pthread_setcancelstate*() must be used to disable cancellation for the duration of any calls that the signal handler makes which are cancellation points. However, the standard does not permit strictly conforming applications to call *pthread_setcancelstate*() from a signal handler since it is not currently required to be async-signal-safe. On implementations where *pthread_setcancelstate*() is not async-signal-safe, alternatives are to ensure either that the corresponding signals are blocked during execution of functions that are not async-cancel-safe or that cancellation is disabled during times when those signals could be delivered. Implementations are strongly encouraged to make *pthread_setcancelstate*() async-signal-safe.

RATIONALE

The *pthread_setcancelstate*() and *pthread_setcanceltype*() functions control the points at which a thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, some rules need to be followed.

An object can be considered to be a generalization of a procedure. It is a set of procedures and global variables written as a unit and called by clients not known by the object. Objects may depend on other objects.

First, cancelability should only be disabled on entry to an object, never explicitly enabled. On exit from an object, the cancelability state should always be restored to its value on entry to the object.

This follows from a modularity argument: if the client of an object (or the client of an object that uses that object) has disabled cancelability, it is because the client does not want to be concerned about cleaning up if the thread is canceled while executing some sequence of actions. If an object is called in such a state and it enables cancelability and a cancellation request is pending for that thread, then the thread is canceled, contrary to the wish of the client that disabled.

Second, the cancelability type may be explicitly set to either *deferred* or *asynchronous* upon entry to an object. But as with the cancelability state, on exit from an object the cancelability type should always be restored to its value on entry to the object.

Finally, only functions that are cancel-safe may be called from a thread that is asynchronously cancelable.

FUTURE DIRECTIONS

The *pthread_setcancelstate*() function may be added to the table of async-signal-safe functions in *Section 2.4.3*, *Signal Actions*.

SEE ALSO

pthread_cancel( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_setconcurrency — set the level of concurrency

SYNOPSIS

```
#include <pthread.h>

int pthread_setconcurrency(int new_level);
```

DESCRIPTION

Refer to *pthread_getconcurrency*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_setschedparam — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

```
#include <pthread.h>

int pthread_setschedparam(pthread_t thread, int policy,
   const struct sched_param *param);
```

DESCRIPTION

Refer to *pthread_getschedparam*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_setschedprio — dynamic thread scheduling parameters access (**REALTIME THREADS**)

SYNOPSIS

#include <pthread.h>

int pthread_setschedprio(pthread_t *thread*, int *prio*);

DESCRIPTION

The *pthread_setschedprio*() function shall set the scheduling priority for the thread whose thread ID is given by *thread* to the value given by *prio*. See *Scheduling Policies* for a description on how this function call affects the ordering of the thread in the thread list for its new priority.

If the *pthread_setschedprio*() function fails, the scheduling priority of the target thread shall not be changed.

RETURN VALUE

If successful, the *pthread_setschedprio*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_setschedprio*() function may fail if:

EINVAL
The value of *prio* is invalid for the scheduling policy of the specified thread.

EPERM
The caller does not have appropriate privileges to set the scheduling priority of the specified thread.

The *pthread_setschedprio*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *pthread_setschedprio*() function provides a way for an application to temporarily raise its priority and then lower it again, without having the undesired side-effect of yielding to other threads of the same priority. This is necessary if the application is to implement its own strategies for bounding priority inversion, such as priority inheritance or priority ceilings. This capability is especially important if the implementation does not support the Thread Priority Protection or Thread Priority Inheritance options, but even if those options are supported it is needed if the application is to bound priority inheritance for other resources, such as semaphores.

The standard developers considered that while it might be preferable conceptually to solve this problem by modifying the specification of *pthread_setschedparam*(), it was too late to make such a change, as there may be implementations that would need to be changed. Therefore, this new function was introduced.

If an implementation detects use of a thread ID after the end of its lifetime, it is recommended that the function should fail and report an **[ESRCH]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

Scheduling Policies, *pthread_getschedparam*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_setspecific — thread-specific data management

SYNOPSIS

#include <pthread.h>

int pthread_setspecific(pthread_key_t *key*, const void **value*);

DESCRIPTION

Refer to *pthread_getspecific*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_sigmask, sigprocmask — examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int pthread_sigmask(int how, const sigset_t *restrict set,
   sigset_t *restrict oset);
int sigprocmask(int how, const sigset_t *restrict set,
   sigset_t *restrict oset);
```

DESCRIPTION

The *pthread_sigmask*() function shall examine or change (or both) the calling thread's signal mask, regardless of the number of threads in the process. The function shall be equivalent to *sigprocmask*(), without the restriction that the call be made in a single-threaded process.

In a single-threaded process, the *sigprocmask*() function shall examine or change (or both) the signal mask of the calling thread.

If the argument *set* is not a null pointer, it points to a set of signals to be used to change the currently blocked set.

The argument *how* indicates the way in which the set is changed, and the application shall ensure it consists of one of the following values:

SIG_BLOCK The resulting set shall be the union of the current set and the signal set pointed to by *set*.

SIG_SETMASK
The resulting set shall be the signal set pointed to by *set*.

SIG_UNBLOCK
The resulting set shall be the intersection of the current set and the complement of the signal set pointed to by *set*.

If the argument *oset* is not a null pointer, the previous mask shall be stored in the location pointed to by *oset*. If *set* is a null pointer, the value of the argument *how* is not significant and the thread's signal mask shall be unchanged; thus the call can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to *sigprocmask*(), at least one of those signals shall be delivered before the call to *sigprocmask*() returns.

It is not possible to block those signals which cannot be ignored. This shall be enforced by the system without causing an error to be indicated.

If any of the SIGFPE, SIGILL, SIGSEGV, or SIGBUS signals are generated while they are blocked, the result is undefined, unless the signal was generated by the action of another process, or by one of the functions *kill*(), *pthread_kill*(), *raise*(), or *sigqueue*().

If *sigprocmask*() fails, the thread's signal mask shall not be changed.

The use of the *sigprocmask*() function is unspecified in a multi-threaded process.

RETURN VALUE

Upon successful completion *pthread_sigmask*() shall return 0; otherwise, it shall return the corresponding error number.

Upon successful completion, *sigprocmask*() shall return 0; otherwise, −1 shall be returned, *errno* shall be set to indicate the error, and the signal mask of the process shall be unchanged.

ERRORS

The *pthread_sigmask*() and *sigprocmask*() functions shall fail if:

EINVAL
The value of the *how* argument is not equal to one of the defined values.

The *pthread_sigmask*() function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

Signaling in a Multi-Threaded Process

This example shows the use of *pthread_sigmask*() in order to deal with signals in a multi-threaded process. It provides a fairly general framework that could be easily adapted/extended.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <signal.h>
#include <string.h>
#include <errno.h>
...
static sigset_t   signal_mask;  /* signals to block         */
int main (int argc, char *argv[])
{
    pthread_t  sig_thr_id;      /* signal handler thread ID */
    int        rc;              /* return code              */
    sigemptyset (&signal_mask);
    sigaddset (&signal_mask, SIGINT);
    sigaddset (&signal_mask, SIGTERM);
    rc = pthread_sigmask (SIG_BLOCK, &signal_mask, NULL);
    if (rc != 0) {
        /* handle error */
        ...
    }
    /* any newly created threads inherit the signal mask */
    rc = pthread_create (&sig_thr_id, NULL, signal_thread, NULL);
    if (rc != 0) {
        /* handle error */
        ...
    }
    /* APPLICATION CODE */
    ...
}
void *signal_thread (void *arg)
{
    int       sig_caught;    /* signal caught       */
    int       rc;            /* returned code       */
    rc = sigwait (&signal_mask, &sig_caught);
    if (rc != 0) {
        /* handle error */
    }
    switch (sig_caught)
```

```
        {
        case SIGINT:     /* process SIGINT  */
           ...
           break;
        case SIGTERM:    /* process SIGTERM */
           ...
           break;
        default:         /* should normally not happen */
           fprintf (stderr, "\nUnexpected signal %d\n", sig_caught);
           break;
        }
    }
```

APPLICATION USAGE

None.

RATIONALE

When a thread's signal mask is changed in a signal-catching function that is installed by *sigaction*(), the restoration of the signal mask on return from the signal-catching function overrides that change (see *sigaction*()). If the signal-catching function was installed with *signal*(), it is unspecified whether this occurs.

See *kill*() for a discussion of the requirement on delivery of signals.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *kill*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigqueue*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_spin_destroy, pthread_spin_init — destroy or initialize a spin lock object

SYNOPSIS

#include <pthread.h>

int pthread_spin_destroy(pthread_spinlock_t **lock*);
int pthread_spin_init(pthread_spinlock_t **lock*, int *pshared*);

DESCRIPTION

The *pthread_spin_destroy*() function shall destroy the spin lock referenced by *lock* and release any resources used by the lock. The effect of subsequent use of the lock is undefined until the lock is reinitialized by another call to *pthread_spin_init*(). The results are undefined if *pthread_spin_destroy*() is called when a thread holds the lock, or if this function is called with an uninitialized thread spin lock.

The *pthread_spin_init*() function shall allocate any resources required to use the spin lock referenced by *lock* and initialize the lock to an unlocked state.

If the Thread Process-Shared Synchronization option is supported and the value of *pshared* is PTHREAD_PROCESS_SHARED, the implementation shall permit the spin lock to be operated upon by any thread that has access to the memory where the spin lock is allocated, even if it is allocated in memory that is shared by multiple processes.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

The results are undefined if *pthread_spin_init*() is called specifying an already initialized spin lock. The results are undefined if a spin lock is used without first being initialized.

If the *pthread_spin_init*() function fails, the lock is not initialized and the contents of *lock* are undefined.

Only the object referenced by *lock* may be used for performing synchronization.

The result of referring to copies of that object in calls to *pthread_spin_destroy*(), *pthread_spin_lock*(), *pthread_spin_trylock*(), or *pthread_spin_unlock*() is undefined.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_spin_init*() function shall fail if:

EAGAIN
The system lacks the necessary resources to initialize another spin lock.

ENOMEM
Insufficient memory exists to initialize the lock.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_destroy*() does not refer to an initialized spin lock object, it is recommended that the function should fail and report an

[EINVAL] error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_destroy*() or *pthread_spin_init*() refers to a locked spin lock object, or detects that the value specified by the *lock* argument to *pthread_spin_init*() refers to an already initialized spin lock object, it is recommended that the function should fail and report an **[EBUSY]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_lock( ), *pthread_spin_unlock*( )

The Base Definitions volume of POSIX.1-2017, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_spin_lock, pthread_spin_trylock — lock a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_lock(pthread_spinlock_t *lock);
int pthread_spin_trylock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_lock*() function shall lock the spin lock referenced by *lock*. The calling thread shall acquire the lock if it is not held by another thread. Otherwise, the thread shall spin (that is, shall not return from the *pthread_spin_lock*() call) until the lock becomes available. The results are undefined if the calling thread holds the lock at the time the call is made. The *pthread_spin_trylock*() function shall lock the spin lock referenced by *lock* if it is not held by any thread. Otherwise, the function shall fail.

The results are undefined if any of these functions is called with an uninitialized spin lock.

RETURN VALUE

Upon successful completion, these functions shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

The *pthread_spin_lock*() function may fail if:

EDEADLK
A deadlock condition was detected.

The *pthread_spin_trylock*() function shall fail if:

EBUSY
A thread currently holds the lock.

These functions shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using this function may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock*() or *pthread_spin_trylock*() does not refer to an initialized spin lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_lock*() refers to a spin lock object for which the calling thread already holds the lock, it is recommended that the function should fail and report an **[EDEADLK]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy( ), *pthread_spin_unlock*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_spin_unlock — unlock a spin lock object

SYNOPSIS

```
#include <pthread.h>

int pthread_spin_unlock(pthread_spinlock_t *lock);
```

DESCRIPTION

The *pthread_spin_unlock*() function shall release the spin lock referenced by *lock* which was locked via the *pthread_spin_lock*() or *pthread_spin_trylock*() functions.

The results are undefined if the lock is not held by the calling thread.

If there are threads spinning on the lock when *pthread_spin_unlock*() is called, the lock becomes available and an unspecified spinning thread shall acquire the lock.

The results are undefined if this function is called with an uninitialized thread spin lock.

RETURN VALUE

Upon successful completion, the *pthread_spin_unlock*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

This function shall not return an error code of **[EINTR]**.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_unlock*() does not refer to an initialized spin lock object, it is recommended that the function should fail and report an **[EINVAL]** error.

If an implementation detects that the value specified by the *lock* argument to *pthread_spin_unlock*() refers to a spin lock object for which the current thread does not hold the lock, it is recommended that the function should fail and report an **[EPERM]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_spin_destroy( ), *pthread_spin_lock*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<pthread.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pthread_testcancel — set cancelability state

SYNOPSIS

```
#include <pthread.h>

void pthread_testcancel(void);
```

DESCRIPTION

Refer to *pthread_setcancelstate*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ptsname — get name of the slave pseudo-terminal device

SYNOPSIS

```
#include <stdlib.h>

char *ptsname(int fildes);
```

DESCRIPTION

The *ptsname*() function shall return the name of the slave pseudo-terminal device associated with a master pseudo-terminal device. The *fildes* argument is a file descriptor that refers to the master device. The *ptsname*() function shall return a pointer to a string containing the pathname of the corresponding slave device.

The *ptsname*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *ptsname*() shall return a pointer to a string which is the name of the pseudo-terminal slave device. Upon failure, *ptsname*() shall return a null pointer and may set *errno*. This could occur if *fildes* is an invalid file descriptor or if the slave device name does not exist in the file system.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ptsname*(). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

ERRORS

The *ptsname*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

ENOTTY
The file associated with the *fildes* argument is not a master pseudo-terminal device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See the RATIONALE section for *posix_openpt*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

grantpt( ), *open*( ), *posix_openpt*( ), *ttyname*( ), *unlockpt*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putc — put a byte on a stream

SYNOPSIS

#include <stdio.h>

int putc(int *c*, FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *putc*() function shall be equivalent to *fputc*(), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fputc*( ).

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *putc*() may treat a *stream* argument with side-effects incorrectly. In particular, *putc*(*c*,**f*++) does not necessarily work correctly. Therefore, use of this function is not recommended in such situations; *fputc*() should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fputc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putc_unlocked — stdio with explicit client locking

SYNOPSIS

#include <stdio.h>

int putc_unlocked(int *c*, FILE **stream*);

DESCRIPTION

Refer to *getc_unlocked*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putchar — put a byte on a stdout stream

SYNOPSIS

#include <stdio.h>

int putchar(int *c*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The function call *putchar*(*c*) shall be equivalent to *putc*(*c*,*stdout*).

RETURN VALUE

Refer to *fputc*( ).

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *putc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putchar_unlocked — stdio with explicit client locking

SYNOPSIS

```
#include <stdio.h>

int putchar_unlocked(int c);
```

DESCRIPTION

Refer to *getc_unlocked*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putenv — change or add a value to an environment

SYNOPSIS

```
#include <stdlib.h>

int putenv(char *string);
```

DESCRIPTION

The *putenv*() function shall use the *string* argument to set environment variable values. The *string* argument should point to a string of the form "*name*=*value*". The *putenv*() function shall make the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* shall become part of the environment, so altering the string shall change the environment.

The *putenv*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *putenv*() shall return 0; otherwise, it shall return a non-zero value and set *errno* to indicate the error.

ERRORS

The *putenv*() function may fail if:

ENOMEM
Insufficient memory was available.

The following sections are informative.

EXAMPLES

Changing the Value of an Environment Variable

The following example changes the value of the *HOME* environment variable to the value **/usr/home**.

```
#include <stdlib.h>
...
static char *var = "HOME=/usr/home";
int ret;

ret = putenv(var);
```

APPLICATION USAGE

The *putenv*() function manipulates the environment pointed to by *environ*, and can be used in conjunction with *getenv*().

See *exec*() for restrictions on changing the environment in multi-threaded applications.

This routine may use *malloc*() to enlarge the environment.

A potential error is to call *putenv*() with an automatic variable as the argument, then return from the calling function while *string* is still part of the environment.

Although the space used by *string* is no longer used once a new string which defines *name* is passed to *putenv*(), if any thread in the application has used *getenv*() to retrieve a pointer to this variable, it should not be freed by calling *free*(). If the changed environment variable is one known by the system (such as the locale environment variables) the application should never free the buffer used by earlier calls to *putenv*() for the same variable.

The *setenv*() function is preferred over this function. One reason is that *putenv*() is optional and therefore less portable. Another is that using *putenv*() can slow down environment searches, as explained in the

RATIONALE section for *getenv*( ).

RATIONALE

Refer to the RATIONALE section in *setenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *free*( ), *getenv*( ), *malloc*( ), *setenv*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putmsg, putpmsg — send a message on a STREAM (**STREAMS**)

SYNOPSIS

```
#include <stropts.h>

int putmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int flags);
int putpmsg(int fildes, const struct strbuf *ctlptr,
    const struct strbuf *dataptr, int band, int flags);
```

DESCRIPTION

The *putmsg*() function shall create a message from a process buffer(s) and send the message to a STREAMS file. The message may contain either a data part, a control part, or both. The data and control parts are distinguished by placement in separate buffers, as described below. The semantics of each part are defined by the STREAMS module that receives the message.

The *putpmsg*() function is equivalent to *putmsg*(), except that the process can send messages in different priority bands. Except where noted, all requirements on *putmsg*() also pertain to *putpmsg*().

The *fildes* argument specifies a file descriptor referencing an open STREAM. The *ctlptr* and *dataptr* arguments each point to a **strbuf** structure.

The *ctlptr* argument points to the structure describing the control part, if any, to be included in the message. The *buf* member in the **strbuf** structure points to the buffer where the control information resides, and the *len* member indicates the number of bytes to be sent. The *maxlen* member is not used by *putmsg*(). In a similar manner, the argument *dataptr* specifies the data, if any, to be included in the message. The *flags* argument indicates what type of message should be sent and is described further below.

To send the data part of a message, the application shall ensure that *dataptr* is not a null pointer and the *len* member of *dataptr* is 0 or greater. To send the control part of a message, the application shall ensure that the corresponding values are set for *ctlptr*. No data (control) part shall be sent if either *dataptr*(*ctlptr*) is a null pointer or the *len* member of *dataptr*(*ctlptr*) is set to −1.

For *putmsg*(), if a control part is specified and *flags* is set to RS_HIPRI, a high priority message shall be sent. If no control part is specified, and *flags* is set to RS_HIPRI, *putmsg*() shall fail and set *errno* to **[EINVAL]**. If *flags* is set to 0, a normal message (priority band equal to 0) shall be sent. If a control part and data part are not specified and *flags* is set to 0, no message shall be sent and 0 shall be returned.

For *putpmsg*(), the flags are different. The *flags* argument is a bitmask with the following mutually-exclusive flags defined: MSG_HIPRI and MSG_BAND. If *flags* is set to 0, *putpmsg*() shall fail and set *errno* to **[EINVAL]**. If a control part is specified and *flags* is set to MSG_HIPRI and *band* is set to 0, a high-priority message shall be sent. If *flags* is set to MSG_HIPRI and either no control part is specified or *band* is set to a non-zero value, *putpmsg*() shall fail and set *errno* to **[EINVAL]**. If *flags* is set to MSG_BAND, then a message shall be sent in the priority band specified by *band*. If a control part and data part are not specified and *flags* is set to MSG_BAND, no message shall be sent and 0 shall be returned.

The *putmsg*() function shall block if the STREAM write queue is full due to internal flow control conditions, with the following exceptions:

* For high-priority messages, *putmsg*() shall not block on this condition and continues processing the message.
* For other messages, *putmsg*() shall not block but shall fail when the write queue is full and O_NONBLOCK is set.

The *putmsg*() function shall also block, unless prevented by lack of internal resources, while waiting for the availability of message blocks in the STREAM, regardless of priority or whether O_NONBLOCK has been specified. No partial message shall be sent.

RETURN VALUE

Upon successful completion, *putmsg*() and *putpmsg*() shall return 0; otherwise, they shall return −1 and set *errno* to indicate the error.

ERRORS

The *putmsg*() and *putpmsg*() functions shall fail if:

EAGAIN
A non-priority message was specified, the O_NONBLOCK flag is set, and the STREAM write queue is full due to internal flow control conditions; or buffers could not be allocated for the message that was to be created.

EBADF
fildes is not a valid file descriptor open for writing.

EINTR
A signal was caught during *putmsg*().

EINVAL
An undefined value is specified in *flags*, or *flags* is set to RS_HIPRI or MSG_HIPRI and no control part is supplied, or the STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer, or *flags* is set to MSG_HIPRI and *band* is non-zero (for *putpmsg*() only).

ENOSR
Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.

ENOSTR
A STREAM is not associated with *fildes*.

ENXIO
A hangup condition was generated downstream for the specified STREAM.

EPIPE or **EIO**
The *fildes* argument refers to a STREAMS-based pipe and the other end of the pipe is closed. A SIGPIPE signal is generated for the calling thread.

ERANGE
The size of the data part of the message does not fall within the range specified by the maximum and minimum packet sizes of the topmost STREAM module. This value is also returned if the control part of the message is larger than the maximum configured size of the control part of a message, or if the data part of a message is larger than the maximum configured size of the data part of a message.

In addition, *putmsg*() and *putpmsg*() shall fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of *putmsg*() or *putpmsg*(), but reflects the prior error.

The following sections are informative.

EXAMPLES

Sending a High-Priority Message

The value of *fd* is assumed to refer to an open STREAMS file. This call to *putmsg*() does the following:

1. Creates a high-priority message with a control part and a data part, using the buffers pointed to by *ctrlbuf* and *databuf*, respectively.
2. Sends the message to the STREAMS file identified by *fd*.

```
#include <stropts.h>
```

```
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putmsg(fd, &ctrl, &data, MSG_HIPRI);
```

Using putpmsg( )

This example has the same effect as the previous example. In this example, however, the *putpmsg*() function creates and sends the message to the STREAMS file.

```
#include <stropts.h>
#include <string.h>
...
int fd;
char *ctrlbuf = "This is the control part";
char *databuf = "This is the data part";
struct strbuf ctrl;
struct strbuf data;
int ret;

ctrl.buf = ctrlbuf;
ctrl.len = strlen(ctrlbuf);

data.buf = databuf;
data.len = strlen(databuf);

ret = putpmsg(fd, &ctrl, &data, 0, MSG_HIPRI);
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

The *putmsg*() and *putpmsg*() functions may be removed in a future version.

SEE ALSO

Section 2.6, *STREAMS*, *getmsg*( ), *poll*( ), *read*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<stropts.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

puts — put a string on standard output

SYNOPSIS

#include <stdio.h>

int puts(const char **s*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *puts*() function shall write the string pointed to by *s*, followed by a <newline>, to the standard output stream *stdout*. The terminating null byte shall not be written.

The last data modification and last file status change timestamps of the file shall be marked for update between the successful execution of *puts*() and the next successful completion of a call to *fflush*() or *fclose*() on the same stream or a call to *exit*() or *abort*().

RETURN VALUE

Upon successful completion, *puts*() shall return a non-negative number. Otherwise, it shall return EOF, shall set an error indicator for the stream, and *errno* shall be set to indicate the error.

ERRORS

Refer to *fputc*( ).

The following sections are informative.

EXAMPLES

Printing to Standard Output

The following example gets the current time, converts it to a string using *localtime*() and *asctime*(), and prints it to standard output using *puts*(). It then prints the number of minutes to an event for which it is waiting.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
   minutes_to_event);
...
```

APPLICATION USAGE

The *puts*() function appends a <newline>, while *fputs*() does not.

This volume of POSIX.1-2017 requires that successful completion simply return a non-negative integer. There are at least three known different implementation conventions for this requirement:

* Return a constant value.

* Return the last character written.
* Return the number of bytes written. Note that this implementation convention cannot be adhered to for strings longer than {INT_MAX} bytes as the value would not be representable in the return type of the function. For backwards compatibility, implementations can return the number of bytes for strings of up to {INT_MAX} bytes, and return {INT_MAX} for all longer strings.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *fputs*( ), *putc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pututxline — put an entry into the user accounting database

SYNOPSIS

#include <utmpx.h>

struct utmpx *pututxline(const struct utmpx **utmpx*);

DESCRIPTION

Refer to *endutxent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putwc — put a wide character on a stream

SYNOPSIS

#include <stdio.h>
#include <wchar.h>

wint_t putwc(wchar_t *wc*, FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *putwc*() function shall be equivalent to *fputwc*(), except that if it is implemented as a macro it may evaluate *stream* more than once, so the argument should never be an expression with side-effects.

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since it may be implemented as a macro, *putwc*() may treat a *stream* argument with side-effects incorrectly. In particular, *putwc*(*wc*,**f*++) need not work correctly. Therefore, use of this function is not recommended; *fputwc*() should be used instead.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fputwc*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

putwchar — put a wide character on a stdout stream

SYNOPSIS

#include <wchar.h>

wint_t putwchar(wchar_t *wc*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The function call *putwchar*(*wc*) shall be equivalent to *putwc*(*wc*,*stdout*).

RETURN VALUE

Refer to *fputwc*( ).

ERRORS

Refer to *fputwc*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fputwc*( ), *putwc*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pwrite — write on a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
    off_t offset);
```

DESCRIPTION

Refer to *write*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

qsort — sort a table of data

SYNOPSIS

```
#include <stdlib.h>

void qsort(void *base, size_t nel, size_t width,
   int (*compar)(const void *, const void *));
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *qsort*() function shall sort an array of *nel* objects, the initial element of which is pointed to by *base*. The size of each object, in bytes, is specified by the *width* argument. If the *nel* argument has the value zero, the comparison function pointed to by *compar* shall not be called and no rearrangement shall take place.

The application shall ensure that the comparison function pointed to by *compar* does not alter the contents of the array. The implementation may reorder elements of the array between calls to the comparison function, but shall not alter the contents of any individual element.

When the same objects (consisting of width bytes, irrespective of their current positions in the array) are passed more than once to the comparison function, the results shall be consistent with one another. That is, they shall define a total ordering on the array.

The contents of the array shall be sorted in ascending order according to a comparison function. The *compar* argument is a pointer to the comparison function, which is called with two arguments that point to the elements being compared. The application shall ensure that the function returns an integer less than, equal to, or greater than 0, if the first argument is considered respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is unspecified.

RETURN VALUE

The *qsort*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

RATIONALE

The requirement that each argument (hereafter referred to as *p)* to the comparison function is a pointer to elements of the array implies that for every call, for each argument separately, all of the following expressions are non-zero:

```
((char *)p - (char *)base) % width == 0
(char *)p >= (char *)base
(char *)p < (char *)base + nel * width
```

FUTURE DIRECTIONS

None.

SEE ALSO

alphasort( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

raise — send a signal to the executing process

SYNOPSIS

```
#include <signal.h>

int raise(int sig);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *raise*() function shall send the signal *sig* to the executing thread or process. If a signal handler is called, the *raise*() function shall not return until after the signal handler does.

The effect of the *raise*() function shall be equivalent to calling:

```
pthread_kill(pthread_self(), sig);
```

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, a non-zero value shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *raise*() function shall fail if:

EINVAL
The value of the *sig* argument is an invalid signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term ''thread'' is an extension to the ISO C standard.

FUTURE DIRECTIONS

None.

SEE ALSO

kill( ), *sigaction*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rand, rand_r, srand — pseudo-random number generator

SYNOPSIS

#include <stdlib.h>

int rand(void);
int rand_r(unsigned **seed*);
void srand(unsigned *seed*);

DESCRIPTION

For *rand*() and *srand*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *rand*() function shall compute a sequence of pseudo-random integers in the range [0,{RAND_MAX}] with a period of at least 2^{32}.

The *rand*() function need not be thread-safe.

The *rand_r*() function shall compute a sequence of pseudo-random integers in the range [0,{RAND_MAX}]. (The value of the {RAND_MAX} macro shall be at least 32 767.)

If *rand_r*() is called with the same initial value for the object pointed to by *seed* and that object is not modified between successive returns and calls to *rand_r*(), the same sequence shall be generated.

The *srand*() function uses the argument as a seed for a new sequence of pseudo-random numbers to be returned by subsequent calls to *rand*(). If *srand*() is then called with the same seed value, the sequence of pseudo-random numbers shall be repeated. If *rand*() is called before any calls to *srand*() are made, the same sequence shall be generated as when *srand*() is first called with a seed value of 1.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *rand*() or *srand*().

RETURN VALUE

The *rand*() function shall return the next pseudo-random number in the sequence.

The *rand_r*() function shall return a pseudo-random integer.

The *srand*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Generating a Pseudo-Random Number Sequence

The following example demonstrates how to generate a sequence of pseudo-random numbers.

```
#include <stdio.h>
#include <stdlib.h>
...
    long count, i;
    char *keystr;
    int elementlen, len;
    char c;
...
```

```
/* Initial random number generator. */
   srand(1);

   /* Create keys using only lowercase characters */
   len = 0;
   for (i=0; i<count; i++) {
       while (len < elementlen) {
           c = (char) (rand() % 128);
           if (islower(c))
               keystr[len++] = c;
       }

       keystr[len] = '\0';
       printf("%s Element%0*ld\n", keystr, elementlen, i);
       len = 0;
   }
```

Generating the Same Sequence on Different Machines

The following code defines a pair of functions that could be incorporated into applications wishing to ensure that the same sequence of numbers is generated across different machines.

```
static unsigned long next = 1;
int myrand(void)  /* RAND_MAX assumed to be 32767. */
{
    next = next * 1103515245 + 12345;
    return((unsigned)(next/65536) % 32768);
}

void mysrand(unsigned seed)
{
    next = seed;
}
```

APPLICATION USAGE

The *drand48*() and *random*() functions provide much more elaborate pseudo-random number generators.

The limitations on the amount of state that can be carried between one function call and another mean the *rand_r*() function can never be implemented in a way which satisfies all of the requirements on a pseudo-random number generator.

These functions should be avoided whenever non-trivial requirements (including safety) have to be fulfilled.

RATIONALE

The ISO C standard *rand*() and *srand*() functions allow per-process pseudo-random streams shared by all threads. Those two functions need not change, but there has to be mutual-exclusion that prevents interference between two threads concurrently accessing the random number generator.

With regard to *rand*(), there are two different behaviors that may be wanted in a multi-threaded program:

1. A single per-process sequence of pseudo-random numbers that is shared by all threads that call *rand*()
2. A different sequence of pseudo-random numbers for each thread that calls *rand*()

This is provided by the modified thread-safe function based on whether the seed value is global to the entire process or local to each thread.

This does not address the known deficiencies of the *rand*() function implementations, which have been approached by maintaining more state. In effect, this specifies new thread-safe forms of a deficient function.

FUTURE DIRECTIONS

The *rand_r*() function may be removed in a future version.

SEE ALSO

drand48( ), *initstate*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

random — generate pseudo-random number

SYNOPSIS

```
#include <stdlib.h>

long random(void);
```

DESCRIPTION

Refer to *initstate*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pread, read — read from a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pread(int fildes, void *buf, size_t nbyte, off_t offset);
ssize_t read(int fildes, void *buf, size_t nbyte);
```

DESCRIPTION

The *read*() function shall attempt to read *nbyte* bytes from the file associated with the open file descriptor, *fildes*, into the buffer pointed to by *buf*. The behavior of multiple concurrent reads on the same pipe, FIFO, or terminal device is unspecified.

Before any action described below is taken, and if *nbyte* is zero, the *read*() function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the *read*() function shall return zero and have no other results.

On files that support seeking (for example, a regular file), the *read*() shall start at a position in the file given by the file offset associated with *fildes*. The file offset shall be incremented by the number of bytes actually read.

Files that do not support seeking—for example, terminals—always read from the current position. The value of a file offset associated with such a file is undefined.

No data transfer shall occur past the current end-of-file. If the starting position is at or after the end-of-file, 0 shall be returned. If the file refers to a device special file, the result of subsequent *read*() requests is implementation-defined.

If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

When attempting to read from an empty pipe or FIFO:

* If no process has the pipe open for writing, *read*() shall return 0 to indicate end-of-file.
* If some process has the pipe open for writing and O_NONBLOCK is set, *read*() shall return −1 and set *errno* to **[EAGAIN]**.
* If some process has the pipe open for writing and O_NONBLOCK is clear, *read*() shall block the calling thread until some data is written or the pipe is closed by all processes that had the pipe open for writing.

When attempting to read a file (other than a pipe or FIFO) that supports non-blocking reads and has no data currently available:

* If O_NONBLOCK is set, *read*() shall return −1 and set *errno* to **[EAGAIN]**.
* If O_NONBLOCK is clear, *read*() shall block the calling thread until some data becomes available.
* The use of the O_NONBLOCK flag has no effect if there is some data available.

The *read*() function reads data previously written to a file. If any portion of a regular file prior to the end-of-file has not been written, *read*() shall return bytes with value 0. For example, *lseek*() allows the file offset to be set beyond the end of existing data in the file. If data is later written at this point, subsequent reads in the gap between the previous end of data and the newly written data shall return bytes with value 0 until data is written into the gap.

Upon successful completion, where *nbyte* is greater than 0, *read*() shall mark for update the last data access timestamp of the file, and shall return the number of bytes read. This number shall never be greater than *nbyte*. The value returned may be less than *nbyte* if the number of bytes left in the file is less than *nbyte*, if the *read*() request was interrupted by a signal, or if the file is a pipe or FIFO or special file and has fewer than *nbyte* bytes immediately available for reading. For example, a *read*() from a file associated with a terminal may return one typed line of data.

If a *read*() is interrupted by a signal before it reads any data, it shall return −1 with *errno* set to **[EINTR]**.

If a *read*() is interrupted by a signal after it has successfully read some data, it shall return the number of bytes read.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *read*() shall be equivalent to *recv*() with no flags set.

If the O_DSYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion. If the O_SYNC and O_RSYNC bits have been set, read I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

If *fildes* refers to a shared memory object, the result of the *read*() function is unspecified.

If *fildes* refers to a typed memory object, the result of the *read*() function is unspecified.

A *read*() from a STREAMS file can read data in three different modes: *byte-stream* mode, *message-nondiscard* mode, and *message-discard* mode. The default shall be byte-stream mode. This can be changed using the I_SRDOPT *ioctl*() request, and can be tested with I_GRDOPT *ioctl*(). In byte-stream mode, *read*() shall retrieve data from the STREAM until as many bytes as were requested are transferred, or until there is no more data to be retrieved. Byte-stream mode ignores message boundaries.

In STREAMS message-nondiscard mode, *read*() shall retrieve data until as many bytes as were requested are transferred, or until a message boundary is reached. If *read*() does not retrieve all the data in a message, the remaining data shall be left on the STREAM, and can be retrieved by the next *read*() call. Message-discard mode also retrieves data until as many bytes as were requested are transferred, or a message boundary is reached. However, unread data remaining in a message after the *read*() returns shall be discarded, and shall not be available for a subsequent *read*(), *getmsg*(), or *getpmsg*() call.

How *read*() handles zero-byte STREAMS messages is determined by the current read mode setting. In byte-stream mode, *read*() shall accept data until it has read *nbyte* bytes, or until there is no more data to read, or until a zero-byte message block is encountered. The *read*() function shall then return the number of bytes read, and place the zero-byte message back on the STREAM to be retrieved by the next *read*(), *getmsg*(), or *getpmsg*(). In message-nondiscard mode or message-discard mode, a zero-byte message shall return 0 and the message shall be removed from the STREAM. When a zero-byte message is read as the first message on a STREAM, the message shall be removed from the STREAM and 0 shall be returned, regardless of the read mode.

A *read*() from a STREAMS file shall return the data in the message at the front of the STREAM head read queue, regardless of the priority band of the message.

By default, STREAMs are in control-normal mode, in which a *read*() from a STREAMS file can only process messages that contain a data part but do not contain a control part. The *read*() shall fail if a message containing a control part is encountered at the STREAM head. This default action can be changed by placing the STREAM in either control-data mode or control-discard mode with the I_SRDOPT *ioctl*() command. In control-data mode, *read*() shall convert any control part to data and pass it to the application before passing any data part originally present in the same message. In control-discard mode, *read*() shall discard message control parts but return to the process any data part in the message.

In addition, *read*() shall fail if the STREAM head had processed an asynchronous error before the call. In this case, the value of *errno* shall not reflect the result of *read*(), but reflect the prior error. If a hangup occurs on the STREAM being read, *read*() shall continue to operate normally until the STREAM head read queue is empty. Thereafter, it shall return 0.

The *pread*() function shall be equivalent to *read*(), except that it shall read from a given position in the file without changing the file offset. The first three arguments to *pread*() are the same as *read*() with the addition of a fourth argument *offset* for the desired position inside the file. An attempt to perform a *pread*() on a file that is incapable of seeking shall result in an error.

RETURN VALUE

Upon successful completion, these functions shall return a non-negative integer indicating the number of bytes actually read. Otherwise, the functions shall return −1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EAGAIN
The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

EBADF
The *fildes* argument is not a valid file descriptor open for reading.

EBADMSG
The file is a STREAM file that is set to control-normal mode and the message waiting to be read includes a control part.

EINTR
The read operation was terminated due to the receipt of a signal, and no data was transferred.

EINVAL
The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

EIO The process is a member of a background process group attempting to read from its controlling terminal, and either the calling thread is blocking SIGTTIN or the process is ignoring SIGTTIN or the process group of the process is orphaned. This error may also be generated for implementation-defined reasons.

EISDIR
The *fildes* argument refers to a directory and the implementation does not allow the directory to be read using *read*() or *pread*(). The *readdir*() function should be used instead.

EOVERFLOW
The file is a regular file, *nbyte* is greater than 0, the starting position is before the end-of-file, and the starting position is greater than or equal to the offset maximum established in the open file description associated with *fildes*.

The *pread*() function shall fail if:

EINVAL
The file is a regular file or block special file, and the *offset* argument is negative. The file offset shall remain unchanged.

ESPIPE
The file is incapable of seeking.

The *read*() function shall fail if:

EAGAIN
The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

EAGAIN or **EWOULDBLOCK**
The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the read operation.

ECONNRESET
A read was attempted on a socket and the connection was forcibly closed by its peer.

ENOTCONN
A read was attempted on a socket that is not connected.

ETIMEDOUT
A read was attempted on a socket and a transmission timeout occurred.

These functions may fail if:

EIO A physical I/O error has occurred.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

The following sections are informative.

EXAMPLES

Reading Data into a Buffer

The following example reads data from the file associated with the file descriptor *fd* into the buffer pointed to by *buf*.

```
#include <sys/types.h>
#include <unistd.h>
...
char buf[20];
size_t nbytes;
ssize_t bytes_read;
int fd;
...
nbytes = sizeof(buf);
bytes_read = read(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

This volume of POSIX.1-2017 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as **[EBADF]**, the concept is meaningless since no file is involved. For errors that are detected immediately, such as **[EAGAIN]**, clearly the offset should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

Note that a *read*() of zero bytes does not modify the last data access timestamp. A *read*() that requests more than zero bytes, but returns zero, is required to modify the last data access timestamp.

Implementations are allowed, but not required, to perform error checking for *read*() requests of zero bytes.

Input and Output

The use of I/O with large byte counts has always presented problems. Ideas such as *lread*() and *lwrite*() (using and returning **long**s) were considered at one time. The current solution is to use abstract types on the ISO C standard function to *read*() and *write*(). The abstract types can be declared so that existing functions work, but can also be declared so that larger types can be represented in future implementations. It is presumed that whatever constraints limit the maximum range of **size_t** also limit portable I/O requests to the same range. This volume of POSIX.1-2017 also limits the range further by requiring that the byte count be limited so that a signed return value remains meaningful. Since the return type is also a (signed) abstract type, the byte count can be defined by the implementation to be larger than an **int** can hold.

The standard developers considered adding atomicity requirements to a pipe or FIFO, but recognized that due to the nature of pipes and FIFOs there could be no guarantee of atomicity of reads of {PIPE_BUF} or any other size that would be an aid to applications portability.

This volume of POSIX.1-2017 requires that no action be taken for *read*() or *write*() when *nbyte* is zero. This is not intended to take precedence over detection of errors (such as invalid buffer pointers or file descriptors). This is consistent with the rest of this volume of POSIX.1-2017, but the phrasing here could be misread to require detection of the zero case before any other errors. A value of zero is to be considered a correct value, for which the semantics are a no-op.

I/O is intended to be atomic to ordinary files and pipes and FIFOs. Atomic means that all the bytes from a single operation that started out together end up together, without interleaving from other I/O operations. It is a known attribute of terminals that this is not honored, and terminals are explicitly (and implicitly permanently) excepted, making the behavior unspecified. The behavior for other device types is also left unspecified, but the wording is intended to imply that future standards might choose to specify atomicity (or not).

There were recommendations to add format parameters to *read*() and *write*() in order to handle networked transfers among heterogeneous file system and base hardware types. Such a facility may be required for support by the OSI presentation of layer services. However, it was determined that this should correspond with similar C-language facilities, and that is beyond the scope of this volume of POSIX.1-2017. The concept was suggested to the developers of the ISO C standard for their consideration as a possible area for future work.

In 4.3 BSD, a *read*() or *write*() that is interrupted by a signal before transferring any data does not by default return an **[EINTR]** error, but is restarted. In 4.2 BSD, 4.3 BSD, and the Eighth Edition, there is an additional function, *select*(), whose purpose is to pause until specified activity (data to read, space to write, and so on) is detected on specified file descriptors. It is common in applications written for those systems for *select*() to be used before *read*() in situations (such as keyboard input) where interruption of I/O due to a signal is desired.

The issue of which files or file types are interruptible is considered an implementation design issue. This is often affected primarily by hardware and reliability issues.

There are no references to actions taken following an ''unrecoverable error''. It is considered beyond the scope of this volume of POSIX.1-2017 to describe what happens in the case of hardware errors.

Earlier versions of this standard allowed two very different behaviors with regard to the handling of interrupts. In order to minimize the resulting confusion, it was decided that POSIX.1-2008 should support only one of these behaviors. Historical practice on AT&T-derived systems was to have *read*() and *write*() return −1 and set *errno* to **[EINTR]** when interrupted after some, but not all, of the data requested had been transferred. However, the US Department of Commerce FIPS 151-1 and FIPS 151-2 require the historical BSD behavior, in which *read*() and *write*() return the number of bytes actually transferred before the interrupt. If −1 is returned when any data is transferred, it is difficult to recover from the error on a seekable device and impossible on a non-seekable device. Most new implementations support this behavior. The behavior required by POSIX.1-2008 is to return the number of bytes transferred.

POSIX.1-2008 does not specify when an implementation that buffers *read*()s actually moves the data into the user-supplied buffer, so an implementation may choose to do this at the latest possible moment. Therefore, an interrupt arriving earlier may not cause *read*() to return a partial byte count, but rather to return −1 and set *errno* to **[EINTR]**.

Consideration was also given to combining the two previous options, and setting *errno* to **[EINTR]** while returning a short count. However, not only is there no existing practice that implements this, it is also contradictory to the idea that when *errno* is set, the function responsible shall return −1.

This volume of POSIX.1-2017 intentionally does not specify any *pread*() errors related to pipes, FIFOs, and sockets other than **[ESPIPE]**.

FUTURE DIRECTIONS

None.

SEE ALSO

fcntl( ), *ioctl*( ), *lseek*( ), *open*( ), *pipe*( ), *readv*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<stropts.h>**, **<sys_uio.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

readdir, readdir_r — read a directory

SYNOPSIS

```
#include <dirent.h>

struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *restrict dirp, struct dirent *restrict entry,
    struct dirent **restrict result);
```

DESCRIPTION

The type **DIR**, which is defined in the *<dirent.h>* header, represents a *directory stream*, which is an ordered sequence of all the directory entries in a particular directory. Directory entries represent files; files may be removed from a directory or added to a directory asynchronously to the operation of *readdir*().

The *readdir*() function shall return a pointer to a structure representing the directory entry at the current position in the directory stream specified by the argument *dirp*, and position the directory stream at the next entry. It shall return a null pointer upon reaching the end of the directory stream. The structure **dirent** defined in the *<dirent.h>* header describes a directory entry. The value of the structure's *d_ino* member shall be set to the file serial number of the file named by the *d_name* member. If the *d_name* member names a symbolic link, the value of the *d_ino* member shall be set to the file serial number of the symbolic link itself.

The *readdir*() function shall not return directory entries containing empty names. If entries for dot or dot-dot exist, one entry shall be returned for dot and one entry shall be returned for dot-dot; otherwise, they shall not be returned.

The application shall not modify the structure to which the return value of *readdir*() points, nor any storage areas pointed to by pointers within the structure. The returned pointer, and pointers within the structure, might be invalidated or the structure or the storage areas might be overwritten by a subsequent call to *readdir*() on the same directory stream. They shall not be affected by a call to *readdir*() on a different directory stream. The returned pointer, and pointers within the structure, might also be invalidated if the calling thread is terminated.

If a file is removed from or added to the directory after the most recent call to *opendir*() or *rewinddir*(), whether a subsequent call to *readdir*() returns an entry for that file is unspecified.

The *readdir*() function may buffer several directory entries per actual read operation; *readdir*() shall mark for update the last data access timestamp of the directory each time the directory is actually read.

After a call to *fork*(), either the parent or child (but not both) may continue processing the directory stream using *readdir*(), *rewinddir*(), or *seekdir*(). If both the parent and child processes use these functions, the result is undefined.

The *readdir*() function need not be thread-safe.

Applications wishing to check for error situations should set *errno* to 0 before calling *readdir*(). If *errno* is set to non-zero on return, an error occurred.

The *readdir_r*() function shall initialize the **dirent** structure referenced by *entry* to represent the directory entry at the current position in the directory stream referred to by *dirp*, store a pointer to this structure at the location referenced by *result*, and position the directory stream at the next entry.

The storage pointed to by *entry* shall be large enough for a **dirent** with an array of **char** *d_name* members containing at least {NAME_MAX}+1 elements.

Upon successful return, the pointer returned at **result* shall have the same value as the argument *entry*. Upon reaching the end of the directory stream, this pointer shall have the value NULL.

The *readdir_r*() function shall not return directory entries containing empty names.

If a file is removed from or added to the directory after the most recent call to *opendir*() or *rewinddir*(), whether a subsequent call to *readdir_r*() returns an entry for that file is unspecified.

The *readdir_r*() function may buffer several directory entries per actual read operation; *readdir_r*() shall mark for update the last data access timestamp of the directory each time the directory is actually read.

RETURN VALUE

Upon successful completion, *readdir*() shall return a pointer to an object of type **struct dirent**. When an error is encountered, a null pointer shall be returned and *errno* shall be set to indicate the error. When the end of the directory is encountered, a null pointer shall be returned and *errno* is not changed.

If successful, the *readdir_r*() function shall return zero; otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions shall fail if:

EOVERFLOW
One of the values in the structure to be returned cannot be represented correctly.

These functions may fail if:

EBADF
The *dirp* argument does not refer to an open directory stream.

ENOENT
The current position of the directory stream is invalid.

The following sections are informative.

EXAMPLES

The following sample program searches the current directory for each of the arguments supplied on the command line.

```
#include <dirent.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>

static void lookup(const char *arg)
{
    DIR *dirp;
    struct dirent *dp;

    if ((dirp = opendir(".")) == NULL) {
        perror("couldn't open '.'");
        return;
    }

    do {
        errno = 0;
        if ((dp = readdir(dirp)) != NULL) {
            if (strcmp(dp->d_name, arg) != 0)
                continue;

            (void) printf("found %s\n", arg);
            (void) closedir(dirp);
                return;

        }
    } while (dp != NULL);
```

```
        if (errno != 0)
            perror("error reading directory");
        else
            (void) printf("failed to find %s\n", arg);
        (void) closedir(dirp);
        return;
    }

    int main(int argc, char *argv[])
    {
        int i;
        for (i = 1; i < argc; i++)
            lookup(argv[i]);
        return (0);
    }
```

APPLICATION USAGE

The *readdir*() function should be used in conjunction with *opendir*(), *closedir*(), and *rewinddir*() to examine the contents of the directory.

The *readdir_r*() function is thread-safe and shall return values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call.

RATIONALE

The returned value of *readdir*() merely *represents* a directory entry. No equivalence should be inferred.

Historical implementations of *readdir*() obtain multiple directory entries on a single read operation, which permits subsequent *readdir*() operations to operate from the buffered information. Any wording that required each successful *readdir*() operation to mark the directory last data access timestamp for update would disallow such historical performance-oriented implementations.

When returning a directory entry for the root of a mounted file system, some historical implementations of *readdir*() returned the file serial number of the underlying mount point, rather than of the root of the mounted file system. This behavior is considered to be a bug, since the underlying file serial number has no significance to applications.

Since *readdir*() returns NULL when it detects an error and when the end of the directory is encountered, an application that needs to tell the difference must set *errno* to zero before the call and check it if NULL is returned. Since the function must not change *errno* in the second case and must set it to a non-zero value in the first case, a zero *errno* after a call returning NULL indicates end-of-directory; otherwise, an error.

Routines to deal with this problem more directly were proposed:

```
    int derror (dirp)
    DIR *dirp;

    void clearderr (dirp)
    DIR *dirp;
```

The first would indicate whether an error had occurred, and the second would clear the error indication. The simpler method involving *errno* was adopted instead by requiring that *readdir*() not change *errno* when end-of-directory is encountered.

An error or signal indicating that a directory has changed while open was considered but rejected.

The thread-safe version of the directory reading function returns values in a user-supplied buffer instead of possibly using a static data area that may be overwritten by each call. Either the {NAME_MAX} compile-time constant or the corresponding *pathconf*() option can be used to determine the maximum sizes of returned pathnames.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *dirfd*( ), *exec*, *fdopendir*( ), *fstatat*( ), *rewinddir*( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

readlink, readlinkat — read the contents of a symbolic link

SYNOPSIS

```
#include <unistd.h>

ssize_t readlink(const char *restrict path, char *restrict buf,
   size_t bufsize);

#include <fcntl.h>

ssize_t readlinkat(int fd, const char *restrict path,
   char *restrict buf, size_t bufsize);
```

DESCRIPTION

The *readlink*() function shall place the contents of the symbolic link referred to by *path* in the buffer *buf* which has size *bufsize*. If the number of bytes in the symbolic link is less than *bufsize*, the contents of the remainder of *buf* are unspecified. If the *buf* argument is not large enough to contain the link content, the first *bufsize* bytes shall be placed in *buf*.

If the value of *bufsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

Upon successful completion, *readlink*() shall mark for update the last data access timestamp of the symbolic link.

The *readlinkat*() function shall be equivalent to the *readlink*() function except in the case where *path* specifies a relative path. In this case the symbolic link whose content is read is relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *readlinkat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *readlink*().

RETURN VALUE

Upon successful completion, these functions shall return the count of bytes placed in the buffer. Otherwise, these functions shall return a value of −1, leave the buffer unchanged, and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EACCES
Search permission is denied for a component of the path prefix of *path*.

EINVAL
The *path* argument names a file that is not a symbolic link.

EIO An I/O error occurred while reading from the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *readlinkat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Reading the Name of a Symbolic Link

The following example shows how to read the name of a symbolic link named **/modules/pass1**.

```
#include <unistd.h>

char buf[1024];
ssize_t len;
...
if ((len = readlink("/modules/pass1", buf, sizeof(buf)-1)) != -1)
    buf[len] = '\0';
```

APPLICATION USAGE

Conforming applications should not assume that the returned contents of the symbolic link are null-terminated.

RATIONALE

The type associated with *bufsiz* is a **size_t** in order to be consistent with both the ISO C standard and the definition of *read*(). The behavior specified for *readlink*() when *bufsiz* is zero represents historical practice. For this case, the standard developers considered a change whereby *readlink*() would return the number of non-null bytes contained in the symbolic link with the buffer *buf* remaining unchanged; however, since the **stat** structure member *st_size* value can be used to determine the size of buffer necessary to contain the contents of the symbolic link as returned by *readlink*(), this proposal was rejected, and the historical practice retained.

The purpose of the *readlinkat*() function is to read the content of symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *readlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *readlinkat*() function it can be guaranteed that the symbolic link read is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

fstatat( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

readv — read a vector

SYNOPSIS

#include <sys/uio.h>

ssize_t readv(int *fildes*, const struct iovec **iov*, int *iovcnt*);

DESCRIPTION

The *readv*() function shall be equivalent to *read*(), except as described below. The *readv*() function shall place the input data into the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], ..., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}.

Each *iovec* entry specifies the base address and length of an area in memory where data should be placed. The *readv*() function shall always fill an area completely before proceeding to the next.

Upon successful completion, *readv*() shall mark for update the last data access timestamp of the file.

RETURN VALUE

Refer to *read*( ).

ERRORS

Refer to *read*( ).

In addition, the *readv*() function shall fail if:

EINVAL
The sum of the *iov_len* values in the *iov* array overflowed an **ssize_t**.

The *readv*() function may fail if:

EINVAL
The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

The following sections are informative.

EXAMPLES

Reading Data into an Array

The following example reads data from the file associated with the file descriptor *fd* into the buffers specified by members of the *iov* array.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_read;
int fd;
char buf0[20];
char buf1[30];
char buf2[40];
int iovcnt;
struct iovec iov[3];

iov[0].iov_base = buf0;
iov[0].iov_len = sizeof(buf0);
iov[1].iov_base = buf1;
iov[1].iov_len = sizeof(buf1);
```

```
iov[2].iov_base = buf2;
iov[2].iov_len = sizeof(buf2);
...
iovcnt = sizeof(iov) / sizeof(struct iovec);

bytes_read = readv(fd, iov, iovcnt);
...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *read*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

read( ), *writev*( )

The Base Definitions volume of POSIX.1-2017, **<sys_uio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

realloc — memory reallocator

SYNOPSIS

#include <stdlib.h>

void *realloc(void **ptr*, size_t *size*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *realloc*() function shall deallocate the old object pointed to by *ptr* and return a pointer to a new object that has the size specified by *size*. The contents of the new object shall be the same as that of the old object prior to deallocation, up to the lesser of the new and old sizes. Any bytes in the new object beyond the size of the old object have indeterminate values. If the size of the space requested is zero, the behavior shall be implementation-defined: either a null pointer is returned, or the behavior shall be as if the size were some non-zero value, except that the behavior is undefined if the returned pointer is used to access an object. If the space cannot be allocated, the object shall remain unchanged.

If *ptr* is a null pointer, *realloc*() shall be equivalent to *malloc*() for the specified size.

If *ptr* does not match a pointer returned earlier by *calloc*(), *malloc*(), or *realloc*() or if the space has previously been deallocated by a call to *free*() or *realloc*(), the behavior is undefined.

The order and contiguity of storage allocated by successive calls to *realloc*() is unspecified. The pointer returned if the allocation succeeds shall be suitably aligned so that it may be assigned to a pointer to any type of object and then used to access such an object in the space allocated (until the space is explicitly freed or reallocated). Each such allocation shall yield a pointer to an object disjoint from any other object. The pointer returned shall point to the start (lowest byte address) of the allocated space. If the space cannot be allocated, a null pointer shall be returned.

RETURN VALUE

Upon successful completion, *realloc*() shall return a pointer to the (possibly moved) allocated space. If *size* is 0, either:

* A null pointer shall be returned and, if *ptr* is not a null pointer, *errno* shall be set to an implementation-defined value.
* A pointer to the allocated space shall be returned, and the memory object pointed to by *ptr* shall be freed. The application shall ensure that the pointer is not used to access an object.

If there is not enough available memory, *realloc*() shall return a null pointer and set *errno* to **[ENOMEM]**. If *realloc*() returns a null pointer and *errno* has been set to **[ENOMEM]**, the memory referenced by *ptr* shall not be changed.

ERRORS

The *realloc*() function shall fail if:

ENOMEM
Insufficient memory is available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The description of *realloc*() has been modified from previous versions of this standard to align with the ISO/IEC 9899: 1999 standard. Previous versions explicitly permitted a call to *realloc*(*p, 0) to free the space pointed to by p* and return a null pointer. While this behavior could be interpreted as permitted by this version of the standard, the C language committee have indicated that this interpretation is incorrect. Applications should assume that if *realloc*() returns a null pointer, the space pointed to by *p* has not been freed. Since this could lead to double-frees, implementations should also set *errno* if a null pointer actually indicates a failure, and applications should only free the space if *errno* was changed.

RATIONALE

None.

FUTURE DIRECTIONS

This standard defers to the ISO C standard. While that standard currently has language that might permit *realloc*(*p, 0), where p* is not a null pointer, to free *p* while still returning a null pointer, the committee responsible for that standard is considering clarifying the language to explicitly prohibit that alternative.

SEE ALSO

calloc( ), *free*( ), *malloc*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

realpath — resolve a pathname

SYNOPSIS

```
#include <stdlib.h>

char *realpath(const char *restrict file_name,
   char *restrict resolved_name);
```

DESCRIPTION

The *realpath*() function shall derive, from the pathname pointed to by *file_name*, an absolute pathname that resolves to the same directory entry, whose resolution does not involve **'.'**, **'..'**, or symbolic links. If *resolved_name* is a null pointer, the generated pathname shall be stored as a null-terminated string in a buffer allocated as if by a call to *malloc*(). Otherwise, if {PATH_MAX} is defined as a constant in the *<limits.h>* header, then the generated pathname shall be stored as a null-terminated string, up to a maximum of {PATH_MAX} bytes, in the buffer pointed to by *resolved_name*.

If *resolved_name* is not a null pointer and {PATH_MAX} is not defined as a constant in the *<limits.h>* header, the behavior is undefined.

RETURN VALUE

Upon successful completion, *realpath*() shall return a pointer to the buffer containing the resolved name. Otherwise, *realpath*() shall return a null pointer and set *errno* to indicate the error.

If the *resolved_name* argument is a null pointer, the pointer returned by *realpath*() can be passed to *free*().

If the *resolved_name* argument is not a null pointer and the *realpath*() function fails, the contents of the buffer pointed to by *resolved_name* are undefined.

ERRORS

The *realpath*() function shall fail if:

EACCES
Search permission was denied for a component of the path prefix of *file_name*.

EINVAL
The *file_name* argument is a null pointer.

EIO An error occurred while reading from the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *file_name* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *file_name* does not name an existing file or *file_name* points to an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *file_name* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *realpath*() function may fail if:

EACCES
The *file_name* argument does not begin with a <slash> and none of the symbolic links (if any) processed during pathname resolution of *file_name* had contents that began with a <slash>, and

either search permission was denied for the current directory or read or search permission was denied for a directory above the current directory in the file hierarchy.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *file_name* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

Generating an Absolute Pathname

The following example generates an absolute pathname for the file identified by the *symlinkpath* argument. The generated pathname is stored in the buffer pointed to by *actualpath*.

```
#include <stdlib.h>
...
char *symlinkpath = "/tmp/symlink/file";
char *actualpath;

actualpath = realpath(symlinkpath, NULL);
if (actualpath != NULL)
{
   ... use actualpath ...

   free(actualpath);
}
else
{
   ... handle error ...
}
```

APPLICATION USAGE

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *realpath*(), this is the return value.

RATIONALE

Since *realpath*() has no *length* argument, if {PATH_MAX} is not defined as a constant in *<limits.h>*, applications have no way of determining how large a buffer they need to allocate for it to be safe to pass to *realpath*(). A {PATH_MAX} value obtained from a prior *pathconf*() call is out-of-date by the time *realpath*() is called. Hence the only reliable way to use *realpath*() when {PATH_MAX} is not defined in *<limits.h>* is to pass a null pointer for *resolved_name* so that *realpath*() will allocate a buffer of the necessary size.

FUTURE DIRECTIONS

None.

SEE ALSO

fpathconf( ), *free*( ), *getcwd*( ), *sysconf*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

recv — receive a message from a connected socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t recv(int socket, void *buffer, size_t length, int flags);
```

DESCRIPTION

The *recv*() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connected sockets because it does not permit the application to retrieve the source address of received data.

The *recv*() function takes the following arguments:

socket	Specifies the socket file descriptor.
buffer	Points to a buffer where the message should be stored.
length	Specifies the length in bytes of the buffer pointed to by the *buffer* argument.
flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next *recv*() or similar function shall still return this data.
MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

The *recv*() function shall return the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, *recv*() shall block until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, *recv*() shall fail and set *errno* to **[EAGAIN]** or **[EWOULDBLOCK]**.

RETURN VALUE

Upon successful completion, *recv*() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recv*() shall return 0. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *recv*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EINTR
The *recv*() function was interrupted by a signal that was caught, before any data was available.

EINVAL
The MSG_OOB flag is set and no out-of-band data is available.

ENOTCONN
A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The specified flags are not supported for this socket type or protocol.

ETIMEDOUT
The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recv*() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
The *recv*() function is equivalent to *recvfrom*() with null pointer *address* and *address_len* arguments, and to *read*() if the *socket* argument refers to a socket and the *flags* argument is 0.

The *select*() and *poll*() functions can be used to determine when data is available to be received.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
poll( ), *pselect*( ), *read*( ), *recvmsg*( ), *recvfrom*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

recvfrom — receive a message from a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t recvfrom(int socket, void *restrict buffer, size_t length,
   int flags, struct sockaddr *restrict address,
   socklen_t *restrict address_len);
```

DESCRIPTION

The *recvfrom*() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

The *recvfrom*() function takes the following arguments:

socket	Specifies the socket file descriptor.
buffer	Points to the buffer where the message should be stored.
length	Specifies the length in bytes of the buffer pointed to by the *buffer* argument.
flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

MSG_PEEK	Peeks at an incoming message. The data is treated as unread and the next *recvfrom*() or similar function shall still return this data.
MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

address	A null pointer, or points to a **sockaddr** structure in which the sending address is to be stored. The length and format of the address depend on the address family of the socket.
address_len	Either a null pointer, if *address* is a null pointer, or a pointer to a **socklen_t** object which on input specifies the length of the supplied **sockaddr** structure, and on output specifies the length of the stored address.

The *recvfrom*() function shall return the length of the message written to the buffer pointed to by the *buffer* argument. For message-based sockets, such as SOCK_RAW, SOCK_DGRAM, and SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

Not all protocols provide the source address for messages. If the *address* argument is not a null pointer and the protocol provides the source address of messages, the source address of the received message shall be stored in the **sockaddr** structure pointed to by the *address* argument, and the length of this address shall be stored in the object pointed to by the *address_len* argument.

If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated.

If the *address* argument is not a null pointer and the protocol does not provide the source address of messages, the value stored in the object pointed to by *address* is unspecified.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, *recvfrom*() shall block until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, *recvfrom*() shall fail and set *errno* to **[EAGAIN]** or **[EWOULDBLOCK]**.

RETURN VALUE

Upon successful completion, *recvfrom*() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvfrom*() shall return 0. Otherwise, the function shall return −1 and set *errno* to indicate the error.

ERRORS

The *recvfrom*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EINTR
A signal interrupted *recvfrom*() before any data was available.

EINVAL
The MSG_OOB flag is set and no out-of-band data is available.

ENOTCONN
A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The specified flags are not supported for this socket type.

ETIMEDOUT
The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recvfrom*() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *select*() and *poll*() functions can be used to determine when data is available to be received.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

poll( ), *pselect*( ), *read*( ), *recv*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

recvmsg — receive a message from a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t recvmsg(int socket, struct msghdr *message, int flags);
```

DESCRIPTION

The *recvmsg*() function shall receive a message from a connection-mode or connectionless-mode socket. It is normally used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

The *recvmsg*() function takes the following arguments:

socket	Specifies the socket file descriptor.
message	Points to a **msghdr** structure, containing both the buffer to store the source address and the buffers for the incoming message. The length and format of the address depend on the address family of the socket. The *msg_flags* member is ignored on input, but may contain meaningful values on output.
flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more of the following values:

MSG_OOB	Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
MSG_PEEK	Peeks at the incoming message.
MSG_WAITALL	On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned. The function may return the smaller amount of data if the socket is a message-based socket, if a signal is caught, if the connection is terminated, if MSG_PEEK was specified, or if an error is pending for the socket.

The *recvmsg*() function shall receive messages from unconnected or connected sockets and shall return the length of the message.

The *recvmsg*() function shall return the total length of the message. For message-based sockets, such as SOCK_DGRAM and SOCK_SEQPACKET, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffers, and MSG_PEEK is not set in the *flags* argument, the excess bytes shall be discarded, and MSG_TRUNC shall be set in the *msg_flags* member of the **msghdr** structure. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored. In this case, data shall be returned to the user as soon as it becomes available, and no data shall be discarded.

If the MSG_WAITALL flag is not set, data shall be returned only up to the end of the first message.

If no messages are available at the socket and O_NONBLOCK is not set on the socket's file descriptor, *recvmsg*() shall block until a message arrives. If no messages are available at the socket and O_NONBLOCK is set on the socket's file descriptor, the *recvmsg*() function shall fail and set *errno* to **[EAGAIN]** or **[EWOULDBLOCK]**.

In the **msghdr** structure, the *msg_name* member may be a null pointer if the source address is not required. Otherwise, if the socket is unconnected, the *msg_name* member points to a **sockaddr** structure in which the source address is to be stored, and the *msg_namelen* member on input specifies the length of the supplied **sockaddr** structure and on output specifies the length of the stored address. If the actual length of the address is greater than the length of the supplied **sockaddr** structure, the stored address shall be truncated. If the socket is connected, the *msg_name* and *msg_namelen* members shall be ignored. The *msg_iov* and *msg_iovlen* fields are used to specify where the received data shall be stored. The *msg_iov* member points to an array of **iovec** structures; the *msg_iovlen* member shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Each storage area indicated by *msg_iov* is filled with received data in turn until all of the received data is stored or all of the areas have been filled.

Upon successful completion, the *msg_flags* member of the message header shall be the bitwise-inclusive OR of all of the following flags that indicate conditions detected for the received message:

MSG_EOR	End-of-record was received (if supported by the protocol).
MSG_OOB	Out-of-band data was received.
MSG_TRUNC	Normal data was truncated.
MSG_CTRUNC	Control data was truncated.

RETURN VALUE

Upon successful completion, *recvmsg*() shall return the length of the message in bytes. If no messages are available to be received and the peer has performed an orderly shutdown, *recvmsg*() shall return 0. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *recvmsg*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and no data is waiting to be received; or MSG_OOB is set and no out-of-band data is available and either the socket's file descriptor is marked O_NONBLOCK or the socket does not support blocking to await out-of-band data.

EBADF
The *socket* argument is not a valid open file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EINTR
This function was interrupted by a signal before any data was available.

EINVAL
The sum of the *iov_len* values overflows a **ssize_t**, or the MSG_OOB flag is set and no out-of-band data is available.

EMSGSIZE
The *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than or equal to 0, or is greater than {IOV_MAX}.

ENOTCONN
A receive is attempted on a connection-mode socket that is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The specified flags are not supported for this socket type.

ETIMEDOUT
The connection timed out during connection establishment, or due to a transmission timeout on active connection.

The *recvmsg*() function may fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *select*() and *poll*() functions can be used to determine when data is available to be received.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

poll( ), *pselect*( ), *recv*( ), *recvfrom*( ), *send*( ), *sendmsg*( ), *sendto*( ), *shutdown*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

regcomp, regerror, regexec, regfree — regular expression matching

SYNOPSIS

```
#include <regex.h>

int regcomp(regex_t *restrict preg, const char *restrict pattern,
    int cflags);
size_t regerror(int errcode, const regex_t *restrict preg,
    char *restrict errbuf, size_t errbuf_size);
int regexec(const regex_t *restrict preg, const char *restrict string,
    size_t nmatch, regmatch_t pmatch[restrict], int eflags);
void regfree(regex_t *preg);
```

DESCRIPTION

These functions interpret *basic* and *extended* regular expressions as described in the Base Definitions volume of POSIX.1-2017, *Chapter 9*, *Regular Expressions*.

The **regex_t** structure is defined in *<regex.h>* and contains at least the following member:

center box tab(!); cB | cB | cB lw(1.25i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ size_t!re_nsub!T{ Number of parenthesized subexpressions. T}

The **regmatch_t** structure is defined in *<regex.h>* and contains at least the following members:

center box tab(!); cB | cB | cB lw(1.25i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ regoff_t!rm_so!T{ Byte offset from start of *string* to start of substring. T} regoff_t!rm_eo!T{ Byte offset from start of *string* of the first character after the end of substring. T}

The *regcomp*() function shall compile the regular expression contained in the string pointed to by the *pattern* argument and place the results in the structure pointed to by *preg*. The *cflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the *<regex.h>* header:

REG_EXTENDED
: Use Extended Regular Expressions.

REG_ICASE
: Ignore case in match (see the Base Definitions volume of POSIX.1-2017, *Chapter 9*, *Regular Expressions*).

REG_NOSUB
: Report only success/fail in *regexec*().

REG_NEWLINE
: Change the handling of <newline> characters, as described in the text.

The default regular expression type for *pattern* is a Basic Regular Expression. The application can specify Extended Regular Expressions using the REG_EXTENDED *cflags* flag.

If the REG_NOSUB flag was not set in *cflags*, then *regcomp*() shall set *re_nsub* to the number of parenthesized subexpressions (delimited by "\(\)" in basic regular expressions or "()" in extended regular expressions) found in *pattern*.

The *regexec*() function compares the null-terminated string specified by *string* with the compiled regular expression *preg* initialized by a previous call to *regcomp*(). If it finds a match, *regexec*() shall return 0; otherwise, it shall return non-zero indicating either no match or an error. The *eflags* argument is the bitwise-inclusive OR of zero or more of the following flags, which are defined in the *<regex.h>* header:

REG_NOTBOL
: The first character of the string pointed to by *string* is not the beginning of the line. Therefore, the <circumflex> character ('^'), when taken as a special character, shall not match the beginning of *string*.

REG_NOTEOL The last character of the string pointed to by *string* is not the end of the line. Therefore, the <dollar-sign> (**'$'**), when taken as a special character, shall not match the end of *string*.

If *nmatch* is 0 or REG_NOSUB was set in the *cflags* argument to *regcomp*(), then *regexec*() shall ignore the *pmatch* argument. Otherwise, the application shall ensure that the *pmatch* argument points to an array with at least *nmatch* elements, and *regexec*() shall fill in the elements of that array with offsets of the substrings of *string* that correspond to the parenthesized subexpressions of *pattern*: *pmatch*[*i*].*rm_so* shall be the byte offset of the beginning and *pmatch*[*i*].*rm_eo* shall be one greater than the byte offset of the end of substring *i*. (Subexpression *i* begins at the *i*th matched open parenthesis, counting from 1.) Offsets in *pmatch*[0] identify the substring that corresponds to the entire regular expression. Unused elements of *pmatch* up to *pmatch*[*nmatch*−1] shall be filled with −1. If there are more than *nmatch* subexpressions in *pattern* (*pattern* itself counts as a subexpression), then *regexec*() shall still do the match, but shall record only the first *nmatch* substrings.

When matching a basic or extended regular expression, any given parenthesized subexpression of *pattern* might participate in the match of several different substrings of *string*, or it might not match any substring even though the pattern as a whole did match. The following rules shall be used to determine which substrings to report in *pmatch* when matching regular expressions:

1. If subexpression *i* in a regular expression is not contained within another subexpression, and it participated in the match several times, then the byte offsets in *pmatch*[*i*] shall delimit the last such match.

2. If subexpression *i* is not contained within another subexpression, and it did not participate in an otherwise successful match, the byte offsets in *pmatch*[*i*] shall be −1. A subexpression does not participate in the match when:

   '*' or **"\{\}"** appears immediately after the subexpression in a basic regular expression, or **'*'**, **'?'**, or **"{ }"** appears immediately after the subexpression in an extended regular expression, and the subexpression did not match (matched 0 times)

   or:

   > **'|'** is used in an extended regular expression to select this subexpression or another, and the other subexpression matched.

3. If subexpression *i* is contained within another subexpression *j*, and *i* is not contained within any other subexpression that is contained within *j*, and a match of subexpression *j* is reported in *pmatch*[*j*], then the match or non-match of subexpression *i* reported in *pmatch*[*i*] shall be as described in 1. and 2. above, but within the substring reported in *pmatch*[*j*] rather than the whole string. The offsets in *pmatch*[*i*] are still relative to the start of *string*.

4. If subexpression *i* is contained in subexpression *j*, and the byte offsets in *pmatch*[*j*] are −1, then the pointers in *pmatch*[*i*] shall also be −1.

5. If subexpression *i* matched a zero-length string, then both byte offsets in *pmatch*[*i*] shall be the byte offset of the character or null terminator immediately following the zero-length string.

If, when *regexec*() is called, the locale is different from when the regular expression was compiled, the result is undefined.

If REG_NEWLINE is not set in *cflags*, then a <newline> in *pattern* or *string* shall be treated as an ordinary character. If REG_NEWLINE is set, then <newline> shall be treated as an ordinary character except as follows:

1. A <newline> in *string* shall not be matched by a <period> outside a bracket expression or by any form of a non-matching list (see the Base Definitions volume of POSIX.1-2017, *Chapter 9*, *Regular Expressions*).

2. A <circumflex> (**'^'**) in *pattern*, when used to specify expression anchoring (see the Base Definitions volume of POSIX.1-2017, *Section 9.3.8*, *BRE Expression Anchoring*), shall match the zero-length string immediately after a <newline> in *string*, regardless of the setting of REG_NOTBOL.

3. A <dollar-sign> (**'$'**) in *pattern*, when used to specify expression anchoring, shall match the zero-length string immediately before a <newline> in *string*, regardless of the setting of REG_NOTEOL.

The *regfree*() function frees any memory allocated by *regcomp*() associated with *preg*.

The following constants are defined as the minimum set of error return values, although other errors listed as implementation extensions in *<regex.h>* are possible:

REG_BADBR	Content of **"\{\}"** invalid: not a number, number too large, more than two numbers, first larger than second.
REG_BADPAT	Invalid regular expression.
REG_BADRPT	**'?'**, **'*'**, or **'+'** not preceded by valid regular expression.
REG_EBRACE	**"\{\}"** imbalance.
REG_EBRACK	**"[]"** imbalance.
REG_ECOLLATE	Invalid collating element referenced.
REG_ECTYPE	Invalid character class type referenced.
REG_EESCAPE	Trailing <backslash> character in pattern.
REG_EPAREN	**"\(\)"** or **"()"** imbalance.
REG_ERANGE	Invalid endpoint in range expression.
REG_ESPACE	Out of memory.
REG_ESUBREG	Number in **"\digit"** invalid or in error.
REG_NOMATCH	*regexec*() failed to match.

If more than one error occurs in processing a function call, any one of the possible constants may be returned, as the order of detection is unspecified.

The *regerror*() function provides a mapping from error codes returned by *regcomp*() and *regexec*() to unspecified printable strings. It generates a string corresponding to the value of the *errcode* argument, which the application shall ensure is the last non-zero value returned by *regcomp*() or *regexec*() with the given value of *preg*. If *errcode* is not such a value, the content of the generated string is unspecified.

If *preg* is a null pointer, but *errcode* is a value returned by a previous call to *regexec*() or *regcomp*(), the *regerror*() still generates an error string corresponding to the value of *errcode*, but it might not be as detailed under some implementations.

If the *errbuf_size* argument is not 0, *regerror*() shall place the generated string into the buffer of size *errbuf_size* bytes pointed to by *errbuf*. If the string (including the terminating null) cannot fit in the buffer, *regerror*() shall truncate the string and null-terminate the result.

If *errbuf_size* is 0, *regerror*() shall ignore the *errbuf* argument, and return the size of the buffer needed to hold the generated string.

If the *preg* argument to *regexec*() or *regfree*() is not a compiled regular expression returned by *regcomp*(), the result is undefined. A *preg* is no longer treated as a compiled regular expression after it is given to *regfree*().

RETURN VALUE

Upon successful completion, the *regcomp*() function shall return 0. Otherwise, it shall return an integer value indicating an error as described in *<regex.h>*, and the content of *preg* is undefined. If a code is returned, the interpretation shall be as given in *<regex.h>*.

If *regcomp*() detects an invalid RE, it may return REG_BADPAT, or it may return one of the error codes that more precisely describes the error.

Upon successful completion, the *regexec*() function shall return 0. Otherwise, it shall return REG_NOMATCH to indicate no match.

Upon successful completion, the *regerror*() function shall return the number of bytes needed to hold the entire generated string, including the null termination. If the return value is greater than *errbuf_size*, the string returned in the buffer pointed to by *errbuf* has been truncated.

The *regfree*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

```
#include <regex.h>

/*
 * Match string against the extended regular expression in
 * pattern, treating errors as no match.
 *
 * Return 1 for match, 0 for no match.
 */

int
match(const char *string, char *pattern)
{
    int    status;
    regex_t    re;

    if (regcomp(&re, pattern, REG_EXTENDED|REG_NOSUB) != 0) {
        return(0);      /* Report error. */
    }
    status = regexec(&re, string, (size_t) 0, NULL, 0);
    regfree(&re);
    if (status != 0) {
        return(0);      /* Report error. */
    }
    return(1);
}
```

The following demonstrates how the REG_NOTBOL flag could be used with *regexec*() to find all substrings in a line that match a pattern supplied by a user. (For simplicity of the example, very little error checking is done.)

```
(void) regcomp (&re, pattern, 0);
/* This call to regexec() finds the first match on the line. */
error = regexec (&re, &buffer[0], 1, &pm, 0);
while (error == 0) {  /* While matches found. */
    /* Substring found between pm.rm_so and pm.rm_eo. */
    /* This call to regexec() finds the next match. */
    error = regexec (&re, buffer + pm.rm_eo, 1, &pm, REG_NOTBOL);
}
```

APPLICATION USAGE

An application could use:

```
regerror(code,preg,(char *)NULL,(size_t)0)
```

to find out how big a buffer is needed for the generated string, *malloc*() a buffer to hold the string, and then call *regerror*() again to get the string. Alternatively, it could allocate a fixed, static buffer that is big enough to hold most strings, and then use *malloc*() to allocate a larger buffer if it finds that this is too small.

To match a pattern as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13*, *Pattern Matching Notation*, use the *fnmatch*() function.

RATIONALE

The *regexec*() function must fill in all *nmatch* elements of *pmatch*, where *nmatch* and *pmatch* are supplied by the application, even if some elements of *pmatch* do not correspond to subexpressions in *pattern*. The application developer should note that there is probably no reason for using a value of *nmatch* that is larger than *preg*−>*re_nsub*+1.

The REG_NEWLINE flag supports a use of RE matching that is needed in some applications like text editors. In such applications, the user supplies an RE asking the application to find a line that matches the given expression. An anchor in such an RE anchors at the beginning or end of any line. Such an application can pass a sequence of <newline>-separated lines to *regexec*() as a single long string and specify REG_NEWLINE to *regcomp*() to get the desired behavior. The application must ensure that there are no explicit <newline> characters in *pattern* if it wants to ensure that any match occurs entirely within a single line.

The REG_NEWLINE flag affects the behavior of *regexec*(), but it is in the *cflags* parameter to *regcomp*() to allow flexibility of implementation. Some implementations will want to generate the same compiled RE in *regcomp*() regardless of the setting of REG_NEWLINE and have *regexec*() handle anchors differently based on the setting of the flag. Other implementations will generate different compiled REs based on the REG_NEWLINE.

The REG_ICASE flag supports the operations taken by the *grep* **−i** option and the historical implementations of *ex* and *vi*. Including this flag will make it easier for application code to be written that does the same thing as these utilities.

The substrings reported in *pmatch*[ ] are defined using offsets from the start of the string rather than pointers. This allows type-safe access to both constant and non-constant strings.

The type **regoff_t** is used for the elements of *pmatch*[ ] to ensure that the application can represent large arrays in memory (important for an application conforming to the Shell and Utilities volume of POSIX.1-2017).

The 1992 edition of this standard required **regoff_t** to be at least as wide as **off_t**, to facilitate future extensions in which the string to be searched is taken from a file. However, these future extensions have not appeared. The requirement rules out popular implementations with 32-bit **regoff_t** and 64-bit **off_t**, so it has been removed.

The standard developers rejected the inclusion of a *regsub*() function that would be used to do substitutions for a matched RE. While such a routine would be useful to some applications, its utility would be much more limited than the matching function described here. Both RE parsing and substitution are possible to implement without support other than that required by the ISO C standard, but matching is much more complex than substituting. The only difficult part of substitution, given the information supplied by *regexec*(), is finding the next character in a string when there can be multi-byte characters. That is a much larger issue, and one that needs a more general solution.

The *errno* variable has not been used for error returns to avoid filling the *errno* name space for this feature.

The interface is defined so that the matched substrings *rm_sp* and *rm_ep* are in a separate **regmatch_t** structure instead of in **regex_t**. This allows a single compiled RE to be used simultaneously in several contexts; in *main*() and a signal handler, perhaps, or in multiple threads of lightweight processes. (The *preg* argument to *regexec*() is declared with type **const**, so the implementation is not permitted to use the structure to store intermediate results.) It also allows an application to request an arbitrary number of substrings from an RE. The number of subexpressions in the RE is reported in *re_nsub* in *preg*. With this change to *regexec*(), consideration was given to dropping the REG_NOSUB flag since the user can now specify this with a zero *nmatch* argument to *regexec*(). However, keeping REG_NOSUB allows an implementation to use a different (perhaps more efficient) algorithm if it knows in *regcomp*() that no subexpressions need be reported. The implementation is only required to fill in *pmatch* if *nmatch* is not zero and if REG_NOSUB is not specified. Note that the **size_t** type, as defined in the ISO C standard, is unsigned, so the description of *regexec*() does not need to address negative values of *nmatch*.

REG_NOTBOL was added to allow an application to do repeated searches for the same pattern in a line. If the pattern contains a <circumflex> character that should match the beginning of a line, then the pattern should only match when matched against the beginning of the line. Without the REG_NOTBOL flag, the application could rewrite the expression for subsequent matches, but in the general case this would require parsing the expression. The need for REG_NOTEOL is not as clear; it was added for symmetry.

The addition of the *regerror*() function addresses the historical need for conforming application programs to have access to error information more than ``Function failed to compile/match your RE for unknown reasons''.

This interface provides for two different methods of dealing with error conditions. The specific error codes (REG_EBRACE, for example), defined in *<regex.h>*, allow an application to recover from an error if it is so able. Many applications, especially those that use patterns supplied by a user, will not try to deal with specific error cases, but will just use *regerror*() to obtain a human-readable error message to present to the user.

The *regerror*() function uses a scheme similar to *confstr*() to deal with the problem of allocating memory to hold the generated string. The scheme used by *strerror*() in the ISO C standard was considered unacceptable since it creates difficulties for multi-threaded applications.

The *preg* argument is provided to *regerror*() to allow an implementation to generate a more descriptive message than would be possible with *errcode* alone. An implementation might, for example, save the character offset of the offending character of the pattern in a field of *preg*, and then include that in the generated message string. The implementation may also ignore *preg*.

A REG_FILENAME flag was considered, but omitted. This flag caused *regexec*() to match patterns as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.13*, *Pattern Matching Notation* instead of REs. This service is now provided by the *fnmatch*() function.

Notice that there is a difference in philosophy between the ISO POSIX-2: 1993 standard and POSIX.1-2008 in how to handle a ``bad'' regular expression. The ISO POSIX-2: 1993 standard says that many bad constructs ``produce undefined results'', or that ``the interpretation is undefined''. POSIX.1-2008, however, says that the interpretation of such REs is unspecified. The term ``undefined'' means that the action by the application is an error, of similar severity to passing a bad pointer to a function.

The *regcomp*() and *regexec*() functions are required to accept any null-terminated string as the *pattern* argument. If the meaning of the string is ``undefined'', the behavior of the function is ``unspecified''. POSIX.1-2008 does not specify how the functions will interpret the pattern; they might return error codes, or they might do pattern matching in some completely unexpected way, but they should not do something like abort the process.

FUTURE DIRECTIONS

None.

SEE ALSO

fnmatch( ), *glob*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 9*, *Regular Expressions*, **<regex.h>**, **<sys_types.h>**

The Shell and Utilities volume of POSIX.1-2017, *Section 2.13*, *Pattern Matching Notation*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

remainder, remainderf, remainderl — remainder function

SYNOPSIS

```
#include <math.h>

double remainder(double x, double y);
float remainderf(float x, float y);
long double remainderl(long double x, long double y);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall return the floating-point remainder $r=x-ny$ when y is non-zero. The value n is the integral value nearest the exact value x/y. When $|n-x/y|=½$, the value n is chosen to be even.

The behavior of *remainder*() shall be independent of the rounding mode.

RETURN VALUE

Upon successful completion, these functions shall return the floating-point remainder $r=x-ny$ when y is non-zero.

On systems that do not support the IEC 60559 Floating-Point option, if y is zero, it is implementation-defined whether a domain error occurs or zero is returned.

If x or y is NaN, a NaN shall be returned.

If x is infinite or y is 0 and the other is non-NaN, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The x argument is ±Inf, or the y argument is ±0 and the other argument is non-NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error

The y argument is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( ), *div*( ), *feclearexcept*( ), *fetestexcept*( ), *ldiv*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

remove — remove a file

SYNOPSIS

```
#include <stdio.h>

int remove(const char *path);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *remove*() function shall cause the file named by the pathname pointed to by *path* to be no longer accessible by that name. A subsequent attempt to open that file using that name shall fail, unless it is created anew.

If *path* does not name a directory, *remove*(*path*) shall be equivalent to *unlink*(*path*).

If *path* names a directory, *remove*(*path*) shall be equivalent to *rmdir*(*path*).

RETURN VALUE

Refer to *rmdir*( ) or *unlink*( ).

ERRORS

Refer to *rmdir*( ) or *unlink*( ).

The following sections are informative.

EXAMPLES

Removing Access to a File

The following example shows how to remove access to a file named **/home/cnd/old_mods**.

```
#include <stdio.h>

int status;
...
status = remove("/home/cnd/old_mods");
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

rmdir( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

remque — remove an element from a queue

SYNOPSIS

```
#include <search.h>

void remque(void *element);
```

DESCRIPTION

Refer to *insque*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

remquo, remquof, remquol — remainder functions

SYNOPSIS

```
#include <math.h>

double remquo(double x, double y, int *quo);
float remquof(float x, float y, int *quo);
long double remquol(long double x, long double y, int *quo);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *remquo*(), *remquof*(), and *remquol*() functions shall compute the same remainder as the *remainder*(), *remainderf*(), and *remainderl*() functions, respectively. In the object pointed to by *quo*, they store a value whose sign is the sign of x/y and whose magnitude is congruent modulo 2^n to the magnitude of the integral quotient of x/y, where n is an implementation-defined integer greater than or equal to 3. If y is zero, the value stored in the object pointed to by *quo* is unspecified.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

These functions shall return *x* REM *y*.

On systems that do not support the IEC 60559 Floating-Point option, if *y* is zero, it is implementation-defined whether a domain error occurs or zero is returned.

If *x* or *y* is NaN, a NaN shall be returned.

If *x* is ±Inf or *y* is zero and the other argument is non-NaN, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is ±Inf, or the *y* argument is ±0 and the other argument is non-NaN.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Domain Error

The *y* argument is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are intended for implementing argument reductions which can exploit a few low-order bits of the quotient. Note that *x* may be so large in magnitude relative to *y* that an exact representation of the quotient is not practical.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *remainder*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rename, renameat — rename file

SYNOPSIS

```
#include <stdio.h>

int rename(const char *old, const char *new);

#include <fcntl.h>

int renameat(int oldfd, const char *old, int newfd,
    const char *new);
```

DESCRIPTION

For *rename*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *rename*() function shall change the name of a file. The *old* argument points to the pathname of the file to be renamed. The *new* argument points to the new pathname of the file. If the *new* argument does not resolve to an existing directory entry for a file of type directory and the *new* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters after all symbolic links have been processed, *rename*() shall fail.

If either the *old* or *new* argument names a symbolic link, *rename*() shall operate on the symbolic link itself, and shall not resolve the last component of the argument. If the *old* argument and the *new* argument resolve to either the same existing directory entry or different directory entries for the same existing file, *rename*() shall return successfully and perform no other action.

If the *old* argument points to the pathname of a file that is not a directory, the *new* argument shall not point to the pathname of a directory. If the link named by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a link named *new* shall remain visible to other threads throughout the renaming operation and refer either to the file referred to by *new* or *old* before the operation began. Write access permission is required for both the directory containing *old* and the directory containing *new*.

If the *old* argument points to the pathname of a directory, the *new* argument shall not point to the pathname of a file that is not a directory. If the directory named by the *new* argument exists, it shall be removed and *old* renamed to *new*. In this case, a link named *new* shall exist throughout the renaming operation and shall refer either to the directory referred to by *new* or *old* before the operation began. If *new* names an existing directory, it shall be required to be an empty directory.

If either *pathname* argument refers to a path whose final component is either dot or dot-dot, *rename*() shall fail.

If the *old* argument points to a pathname of a symbolic link, the symbolic link shall be renamed. If the *new* argument points to a pathname of a symbolic link, the symbolic link shall be removed.

The *old* pathname shall not name an ancestor directory of the *new* pathname. Write access permission is required for the directory containing *old* and the directory containing *new*. If the *old* argument points to the pathname of a directory, write access permission may be required for the directory named by *old*, and, if it exists, the directory named by *new*.

If the link named by the *new* argument exists and the file's link count becomes 0 when it is removed and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before *rename*() returns, but the removal of the file contents shall be postponed until all references to the file are closed.

Upon successful completion, *rename*() shall mark for update the last data modification and last file status change timestamps of the parent directory of each file.

If the *rename*() function fails for any reason other than **[EIO]**, any file named by *new* shall be unaffected.

The *renameat*() function shall be equivalent to the *rename*() function except in the case where either *old* or *new* specifies a relative path. If *old* is a relative path, the file to be renamed is located relative to the directory associated with the file descriptor *oldfd* instead of the current working directory. If *new* is a relative path, the same happens only relative to the directory associated with *newfd*. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *renameat*() is passed the special value AT_FDCWD in the *oldfd* or *newfd* parameter, the current working directory shall be used in the determination of the file for the respective *path* parameter.

RETURN VALUE

Upon successful completion, the *rename*() function shall return 0. Otherwise, it shall return −1, *errno* shall be set to indicate the error, and neither the file named by *old* nor the file named by *new* shall be changed or created.

Upon successful completion, the *renameat*() function shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *rename*() and *renameat*() functions shall fail if:

EACCES
A component of either path prefix denies search permission; or one of the directories containing *old* or *new* denies write permissions; or, write permission is required and is denied for a directory pointed to by the *old* or *new* arguments.

EBUSY
The directory named by *old* or *new* is currently in use by the system or another process, and the implementation considers this an error.

[EEXIST] or [ENOTEMPTY]
The link named by *new* is a directory that is not an empty directory.

EINVAL The *old* pathname names an ancestor directory of the *new* pathname, or either pathname argument contains a final component that is dot or dot-dot.

EIO A physical I/O error has occurred.

EISDIR The *new* argument points to a directory and the *old* argument points to a file that is not a directory.

ELOOP A loop exists in symbolic links encountered during resolution of the *path* argument.

EMLINK The file named by *old* is a directory, and the link count of the parent directory of *new* would exceed {LINK_MAX}.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT The link named by *old* does not name an existing file, a component of the path prefix of *new* does not exist, or either *old* or *new* points to an empty string.

ENOSPC The directory that would contain *new* cannot be extended.

ENOTDIR A component of either path prefix names an existing file that is neither a directory nor a symbolic link to a directory; or the *old* argument names a directory and the *new* argument names a non-directory file; or the *old* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory; or the *old* argument names an existing non-directory file and the *new* argument names a nonexistent file, contains at least one non-<slash> character, and ends with one or more trailing <slash> characters; or the *new* argument names an existing non-directory file, contains at least one non-<slash> character, and ends with one or more trailing <slash> characters.

EPERM or **EACCES**
The S_ISVTX flag is set on the directory containing the file referred to by *old* and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection* with respect to *old*; or *new* refers to an existing file, the S_ISVTX flag is set on the directory containing this file, and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection* with respect to this file.

EROFS The requested operation requires writing in a directory on a read-only file system.

EXDEV The links named by *new* and *old* are on different file systems and the implementation does not support links between file systems.

In addition, the *renameat*() function shall fail if:

EACCES
The access mode of the open file description associated with *oldfd* or *newfd* is not O_SEARCH and the permissions of the directory underlying *oldfd* or *newfd*, respectively, do not permit directory searches.

EBADF
The *old* argument does not specify an absolute path and the *oldfd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching, or the *new* argument does not specify an absolute path and the *newfd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *old* or *new* argument is not an absolute path and *oldfd* or *newfd*, respectively, is a file descriptor associated with a non-directory file.

The *rename*() and *renameat*() functions may fail if:

EBUSY
The file named by the *old* or *new* arguments is a named STREAM.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ETXTBSY
The file named by *new* exists and is the last directory entry to a pure procedure (shared text) file that is being executed.

The following sections are informative.

EXAMPLES

Renaming a File

The following example shows how to rename a file named **/home/cnd/mod1** to **/home/cnd/mod2**.

```
#include <stdio.h>

int status;
...
```

```
status = rename("/home/cnd/mod1", "/home/cnd/mod2");
```

APPLICATION USAGE

Some implementations mark for update the last file status change timestamp of renamed files and some do not. Applications which make use of the last file status change timestamp may behave differently with respect to renamed files unless they are designed to allow for either behavior.

RATIONALE

This *rename*() function is equivalent for regular files to that defined by the ISO C standard. Its inclusion here expands that definition to include actions on directories and specifies behavior when the *new* parameter names a file that already exists. That specification requires that the action of the function be atomic.

One of the reasons for introducing this function was to have a means of renaming directories while permitting implementations to prohibit the use of *link*() and *unlink*() with directories, thus constraining links to directories to those made by *mkdir*().

The specification that if *old* and *new* refer to the same file is intended to guarantee that:

```
rename("x", "x");
```

does not remove the file.

Renaming dot or dot-dot is prohibited in order to prevent cyclical file system paths.

See also the descriptions of **[ENOTEMPTY]** and **[ENAMETOOLONG]** in *rmdir*() and **[EBUSY]** in *unlink*(). For a discussion of **[EXDEV]**, see *link*().

The purpose of the *renameat*() function is to rename files in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *rename*(), resulting in unspecified behavior. By opening file descriptors for the source and target directories and using the *renameat*() function it can be guaranteed that that renamed file is located correctly and the resulting file is in the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

link( ), *rmdir*( ), *symlink*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*, **<fcntl.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rewind — reset the file position indicator in a stream

SYNOPSIS

#include <stdio.h>

void rewind(FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The call:

```
    rewind(stream)
```

shall be equivalent to:

```
    (void) fseek(stream, 0L, SEEK_SET)
```

except that *rewind*() shall also clear the error indicator.

Since *rewind*() does not return a value, an application wishing to detect errors should clear *errno*, then call *rewind*(), and if *errno* is non-zero, assume an error has occurred.

RETURN VALUE

The *rewind*() function shall not return a value.

ERRORS

Refer to *fseek*( ) with the exception of **[EINVAL]** which does not apply.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fseek*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rewinddir — reset the position of a directory stream to the beginning of a directory

SYNOPSIS

```
#include <dirent.h>

void rewinddir(DIR *dirp);
```

DESCRIPTION

The *rewinddir*() function shall reset the position of the directory stream to which *dirp* refers to the beginning of the directory. It shall also cause the directory stream to refer to the current state of the corresponding directory, as a call to *opendir*() would have done. If *dirp* does not refer to a directory stream, the effect is undefined.

After a call to the *fork*() function, either the parent or child (but not both) may continue processing the directory stream using *readdir*(), *rewinddir*(), or *seekdir*(). If both the parent and child processes use these functions, the result is undefined.

RETURN VALUE

The *rewinddir*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *rewinddir*() function should be used in conjunction with *opendir*(), *readdir*(), and *closedir*() to examine the contents of the directory. This method is recommended for portability.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

closedir( ), *fdopendir*( ), *readdir*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rint, rintf, rintl — round-to-nearest integral value

SYNOPSIS

```
#include <math.h>

double rint(double x);
float rintf(float x);
long double rintl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall return the integral value (represented as a **double**) nearest *x* in the direction of the current rounding mode. The current rounding mode is implementation-defined.

If the current rounding mode rounds toward negative infinity, then *rint*() shall be equivalent to *floor*( ). If the current rounding mode rounds toward positive infinity, then *rint*() shall be equivalent to *ceil*( ). If the current rounding mode rounds towards zero, then *rint*() shall be equivalent to *trunc*( ). If the current rounding mode rounds towards nearest, then *rint*() differs from *round*( ) in that halfway cases are rounded to even rather than away from zero.

These functions differ from the *nearbyint*(), *nearbyintf*(), and *nearbyintl*() functions only in that they may raise the inexact floating-point exception if the result differs in value from the argument.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the integer (represented as a double precision number) nearest *x* in the direction of the current rounding mode. The result shall have the same sign as *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

abs( ), *ceil*( ), *feclearexcept*( ), *fetestexcept*( ), *floor*( ), *isnan*( ), *nearbyint*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

rmdir — remove a directory

SYNOPSIS

```
#include <unistd.h>

int rmdir(const char *path);
```

DESCRIPTION

The *rmdir*() function shall remove a directory whose name is given by *path*. The directory shall be removed only if it is an empty directory.

If the directory is the root directory or the current working directory of any process, it is unspecified whether the function succeeds, or whether it shall fail and set *errno* to **[EBUSY]**.

If *path* names a symbolic link, then *rmdir*() shall fail and set *errno* to **[ENOTDIR]**.

If the *path* argument refers to a path whose final component is either dot or dot-dot, *rmdir*() shall fail.

If the directory's link count becomes 0 and no process has the directory open, the space occupied by the directory shall be freed and the directory shall no longer be accessible. If one or more processes have the directory open when the last link is removed, the dot and dot-dot entries, if present, shall be removed before *rmdir*() returns and no new entries may be created in the directory, but the directory shall not be removed until all references to the directory are closed.

If the directory is not an empty directory, *rmdir*() shall fail and set *errno* to **[EEXIST]** or **[ENOTEMPTY]**.

Upon successful completion, *rmdir*() shall mark for update the last data modification and last file status change timestamps of the parent directory.

RETURN VALUE

Upon successful completion, the function *rmdir*() shall return 0. Otherwise, −1 shall be returned, and *errno* set to indicate the error. If −1 is returned, the named directory shall not be changed.

ERRORS

The *rmdir*() function shall fail if:

EACCES
Search permission is denied on a component of the path prefix, or write permission is denied on the parent directory of the directory to be removed.

EBUSY
The directory to be removed is currently in use by the system or some process and the implementation considers this to be an error.

[EEXIST] or [ENOTEMPTY]
The *path* argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.

EINVAL The *path* argument contains a last component that is dot.

EIO A physical I/O error has occurred.

ELOOP A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT A component of *path* does not name an existing file, or the *path* argument names a nonexistent directory or points to an empty string.

ENOTDIR A component of *path* names an existing file that is neither a directory nor a symbolic link to a directory.

[EPERM] or [EACCES]
The S_ISVTX flag is set on the directory containing the file referred to by the *path* argument and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*.

EROFS The directory entry to be removed resides on a read-only file system.

The *rmdir*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Removing a Directory

The following example shows how to remove a directory named **/home/cnd/mod1**.

```
#include <unistd.h>

int status;
...
status = rmdir("/home/cnd/mod1");
```

APPLICATION USAGE

None.

RATIONALE

The *rmdir*() and *rename*() functions originated in 4.2 BSD, and they used **[ENOTEMPTY]** for the condition when the directory to be removed does not exist or *new* already exists. When the 1984 /usr/group standard was published, it contained **[EEXIST]** instead. When these functions were adopted into System V, the 1984 /usr/group standard was used as a reference. Therefore, several existing applications and implementations support/use both forms, and no agreement could be reached on either value. All implementations are required to supply both **[EEXIST]** and **[ENOTEMPTY]** in *<errno.h>* with distinct values, so that applications can use both values in C-language **case** statements.

The meaning of deleting *pathname***/dot** is unclear, because the name of the file (directory) in the parent directory to be removed is not clear, particularly in the presence of multiple links to a directory.

The POSIX.1-1990 standard was silent with regard to the behavior of *rmdir*() when there are multiple hard links to the directory being removed. The requirement to set *errno* to **[EEXIST]** or **[ENOTEMPTY]** clarifies the behavior in this case.

If the current working directory of the process is being removed, that should be an allowed error.

Virtually all existing implementations detect **[ENOTEMPTY]** or the case of dot-dot. The text in *Section 2.3*, *Error Numbers* about returning any one of the possible errors permits that behavior to continue. The **[ELOOP]** error may be returned if more than {SYMLOOP_MAX} symbolic links are encountered during resolution of the *path* argument.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.3, *Error Numbers*, *mkdir*( ), *remove*( ), *rename*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

round, roundf, roundl — round to the nearest integer value in a floating-point format

SYNOPSIS

```
#include <math.h>

double round(double x);
float roundf(float x);
long double roundl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the nearest integer value in floating-point format, rounding halfway cases away from zero, regardless of the current rounding direction.

RETURN VALUE

Upon successful completion, these functions shall return the rounded integer value. The result shall have the same sign as *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

These functions may raise the inexact floating-point exception if the result differs in value from the argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

scalbln, scalblnf, scalblnl, scalbn, scalbnf, scalbnl — compute exponent using FLT_RADIX

SYNOPSIS

#include <math.h>

double scalbln(double *x*, long *n*);
float scalblnf(float *x*, long *n*);
long double scalblnl(long double *x*, long *n*);
double scalbn(double *x*, int *n*);
float scalbnf(float *x*, int *n*);
long double scalbnl(long double *x*, int *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute *x* * FLT_RADIX^n efficiently, not normally by computing FLT_RADIX^n explicitly.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return *x* * FLT_RADIX^n.

If the result would cause overflow, a range error shall occur and these functions shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (according to the sign of *x*) as appropriate for the return type of the function.

If the correct value would cause underflow, and is not representable, a range error may occur, and *scalbln*(), *scalblnf*(), *scalblnl*(), *scalbn*(), *scalbnf*(), and *scalbnl*() shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, LDBL_MIN, DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *n* is 0, *x* shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

ERRORS

These functions shall fail if:

Range Error The result overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno*

shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

These functions are named so as to avoid conflicting with the historical definition of the *scalb*( ) function from the Single UNIX Specification. The difference is that the *scalb*( ) function has a second argument of **double** instead of **int**. The *scalb*( ) function is not part of the ISO C standard. The three functions whose second type is **long** are provided because the factor required to scale from the smallest positive floating-point value to the largest finite one, on many implementations, is too large to represent in the minimum-width **int** format.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

scandir — scan a directory

SYNOPSIS

```
#include <dirent.h>

int scandir(const char *dir, struct dirent ***namelist,
   int (*sel)(const struct dirent *),
   int (*compar)(const struct dirent **, const struct dirent **));
```

DESCRIPTION

Refer to *alphasort*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

scanf — convert formatted input

SYNOPSIS

```
#include <stdio.h>

int scanf(const char *restrict format, ...);
```

DESCRIPTION

Refer to *fscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_get_priority_max, sched_get_priority_min — get priority limits (**REALTIME**)

SYNOPSIS

```
#include <sched.h>

int sched_get_priority_max(int policy);
int sched_get_priority_min(int policy);
```

DESCRIPTION

The *sched_get_priority_max*() and *sched_get_priority_min*() functions shall return the appropriate maximum or minimum, respectively, for the scheduling policy specified by *policy*.

The value of *policy* shall be one of the scheduling policy values defined in *<sched.h>*.

RETURN VALUE

If successful, the *sched_get_priority_max*() and *sched_get_priority_min*() functions shall return the appropriate maximum or minimum values, respectively. If unsuccessful, they shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_get_priority_max*() and *sched_get_priority_min*() functions shall fail if:

EINVAL
The value of the *policy* parameter does not represent a defined scheduling policy.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_setparam*( ), *sched_getscheduler*( ), *sched_rr_get_interval*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_getparam — get scheduling parameters (**REALTIME**)

SYNOPSIS

#include <sched.h>

int sched_getparam(pid_t *pid*, struct sched_param **param*);

DESCRIPTION

The *sched_getparam*() function shall return the scheduling parameters of a process specified by *pid* in the **sched_param** structure pointed to by *param*.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters for the process whose process ID is equal to *pid* shall be returned.

If *pid* is zero, the scheduling parameters for the calling process shall be returned. The behavior of the *sched_getparam*() function is unspecified if the value of *pid* is negative.

RETURN VALUE

Upon successful completion, the *sched_getparam*() function shall return zero. If the call to *sched_getparam*() is unsuccessful, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_getparam*() function shall fail if:

EPERM

The requesting process does not have permission to obtain the scheduling parameters of the specified process.

ESRCH

No process can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getscheduler( ), *sched_setparam*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_getscheduler — get scheduling policy (**REALTIME**)

SYNOPSIS

#include <sched.h>

int sched_getscheduler(pid_t *pid*);

DESCRIPTION

The *sched_getscheduler*() function shall return the scheduling policy of the process specified by *pid*. If the value of *pid* is negative, the behavior of the *sched_getscheduler*() function is unspecified.

The values that can be returned by *sched_getscheduler*() are defined in the *<sched.h>* header.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling policy shall be returned for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling policy shall be returned for the calling process.

RETURN VALUE

Upon successful completion, the *sched_getscheduler*() function shall return the scheduling policy of the specified process. If unsuccessful, the function shall return −1 and set *errno* to indicate the error.

ERRORS

The *sched_getscheduler*() function shall fail if:

EPERM

The requesting process does not have permission to determine the scheduling policy of the specified process.

ESRCH

No process can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_setparam*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_rr_get_interval — get execution time limits (**REALTIME**)

SYNOPSIS

#include <sched.h>

int sched_rr_get_interval(pid_t *pid*, struct timespec **interval*);

DESCRIPTION

The *sched_rr_get_interval*() function shall update the **timespec** structure referenced by the *interval* argument to contain the current execution time limit (that is, time quantum) for the process specified by *pid*. If *pid* is zero, the current execution time limit for the calling process shall be returned.

RETURN VALUE

If successful, the *sched_rr_get_interval*() function shall return zero. Otherwise, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_rr_get_interval*() function shall fail if:

ESRCH

No process can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sched_getparam( ), *sched_get_priority_max*( ), *sched_getscheduler*( ), *sched_setparam*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_setparam — set scheduling parameters (**REALTIME**)

SYNOPSIS

#include <sched.h>

int sched_setparam(pid_t *pid*, const struct sched_param **param*);

DESCRIPTION

The *sched_setparam*() function shall set the scheduling parameters of the process specified by *pid* to the values specified by the **sched_param** structure pointed to by *param*. The value of the *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive priority range for the current scheduling policy of the process specified by *pid*. Higher numerical values for the priority represent higher priorities. If the value of *pid* is negative, the behavior of the *sched_setparam*() function is unspecified.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling parameters shall be set for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling parameters shall be set for the calling process.

The conditions under which one process has permission to change the scheduling parameters of another process are implementation-defined.

Implementations may require the requesting process to have appropriate privileges to set its own scheduling parameters or those of another process.

See *Scheduling Policies* for a description on how this function affects the scheduling of the threads within the target process.

If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy for the underlying kernel-scheduled entities used by the process contention scope threads.

RETURN VALUE

If successful, the *sched_setparam*() function shall return zero.

If the call to *sched_setparam*() is unsuccessful, the priority shall remain unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_setparam*() function shall fail if:

EINVAL

One or more of the requested scheduling parameters is outside the range defined for the scheduling policy of the specified *pid*.

EPERM
The requesting process does not have permission to set the scheduling parameters for the specified process, or does not have appropriate privileges to invoke *sched_setparam*().

ESRCH
No process can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
None.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
Scheduling Policies, *sched_getparam*( ), *sched_getscheduler*( ), *sched_setscheduler*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_setscheduler — set scheduling policy and parameters (**REALTIME**)

SYNOPSIS

```
#include <sched.h>

int sched_setscheduler(pid_t pid, int policy,
   const struct sched_param *param);
```

DESCRIPTION

The *sched_setscheduler*() function shall set the scheduling policy and scheduling parameters of the process specified by *pid* to *policy* and the parameters specified in the **sched_param** structure pointed to by *param*, respectively. The value of the *sched_priority* member in the **sched_param** structure shall be any integer within the inclusive priority range for the scheduling policy specified by *policy*. If the value of *pid* is negative, the behavior of the *sched_setscheduler*() function is unspecified.

The possible values for the *policy* parameter are defined in the *<sched.h>* header.

If a process specified by *pid* exists, and if the calling process has permission, the scheduling policy and scheduling parameters shall be set for the process whose process ID is equal to *pid*.

If *pid* is zero, the scheduling policy and scheduling parameters shall be set for the calling process.

The conditions under which one process has appropriate privileges to change the scheduling parameters of another process are implementation-defined.

Implementations may require that the requesting process have permission to set its own scheduling parameters or those of another process. Additionally, implementation-defined restrictions may apply as to the appropriate privileges required to set the scheduling policy of the process, or the scheduling policy of another process, to a particular value.

The *sched_setscheduler*() function shall be considered successful if it succeeds in setting the scheduling policy and scheduling parameters of the process specified by *pid* to the values specified by *policy* and the structure pointed to by *param*, respectively.

See *Scheduling Policies* for a description on how this function affects the scheduling of the threads within the target process.

If the current scheduling policy for the target process is not SCHED_FIFO, SCHED_RR, or SCHED_SPORADIC, the result is implementation-defined; this case includes the SCHED_OTHER policy.

The specified *sched_ss_repl_period* shall be greater than or equal to the specified *sched_ss_init_budget* for the function to succeed; if it is not, then the function shall fail.

The value of *sched_ss_max_repl* shall be within the inclusive range [1,{SS_REPL_MAX}] for the function to succeed; if not, the function shall fail. It is unspecified whether the *sched_ss_repl_period* and *sched_ss_init_budget* values are stored as provided by this function or are rounded to align with the resolution of the clock being used.

This function is not atomic with respect to other threads in the process. Threads may continue to execute while this function call is in the process of changing the scheduling policy and associated scheduling parameters for the underlying kernel-scheduled entities used by the process contention scope threads.

RETURN VALUE

Upon successful completion, the function shall return the former scheduling policy of the specified process. If the *sched_setscheduler*() function fails to complete successfully, the policy and scheduling parameters shall remain unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sched_setscheduler*() function shall fail if:

EINVAL

The value of the *policy* parameter is invalid, or one or more of the parameters contained in *param* is outside the valid range for the specified scheduling policy.

EPERM

The requesting process does not have permission to set either or both of the scheduling parameters or the scheduling policy of the specified process.

ESRCH

No process can be found corresponding to that specified by *pid*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Scheduling Policies, *sched_getparam*( ), *sched_getscheduler*( ), *sched_setparam*( )

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sched_yield — yield the processor

SYNOPSIS

```
#include <sched.h>

int sched_yield(void);
```

DESCRIPTION

The *sched_yield*() function shall force the running thread to relinquish the processor until it again becomes the head of its thread list. It takes no arguments.

RETURN VALUE

The *sched_yield*() function shall return 0 if it completes successfully; otherwise, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The conceptual model for scheduling semantics in POSIX.1-2008 defines a set of thread lists. This set of thread lists is always present regardless of the scheduling options supported by the system. On a system where the Process Scheduling option is not supported, portable applications should not make any assumptions regarding whether threads from other processes will be on the same thread list.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<sched.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

seed48 — seed a uniformly distributed pseudo-random non-negative long integer generator

SYNOPSIS

```
#include <stdlib.h>

unsigned short *seed48(unsigned short seed16v[3]);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

seekdir — set the position of a directory stream

SYNOPSIS

#include <dirent.h>

void seekdir(DIR **dirp*, long *loc*);

DESCRIPTION

The *seekdir*() function shall set the position of the next *readdir*() operation on the directory stream specified by *dirp* to the position specified by *loc*. The value of *loc* should have been returned from an earlier call to *telldir*() using the same directory stream. The new position reverts to the one associated with the directory stream when *telldir*() was performed.

If the value of *loc* was not obtained from an earlier call to *telldir*(), or if a call to *rewinddir*() occurred between the call to *telldir*() and the call to *seekdir*(), the results of subsequent calls to *readdir*() are unspecified.

RETURN VALUE

The *seekdir*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The original standard developers perceived that there were restrictions on the use of the *seekdir*() and *telldir*() functions related to implementation details, and for that reason these functions need not be supported on all POSIX-conforming systems. They are required on implementations supporting the XSI option.

One of the perceived problems of implementation is that returning to a given point in a directory is quite difficult to describe formally, in spite of its intuitive appeal, when systems that use B-trees, hashing functions, or other similar mechanisms to order their directories are considered. The definition of *seekdir*() and *telldir*() does not specify whether, when using these interfaces, a given directory entry will be seen at all, or more than once.

On systems not supporting these functions, their capability can sometimes be accomplished by saving a filename found by *readdir*() and later using *rewinddir*() and a loop on *readdir*() to relocate the position from which the filename was saved.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir( ), *readdir*( ), *telldir*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

select — synchronous I/O multiplexing

SYNOPSIS

#include <sys/select.h>

int select(int *nfds*, fd_set *restrict *readfds*,
  fd_set *restrict *writefds*, fd_set *restrict *errorfds*,
  struct timeval *restrict *timeout*);

DESCRIPTION

Refer to *pselect*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_close — close a named semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_close(sem_t *sem);
```

DESCRIPTION

The *sem_close*() function shall indicate that the calling process is finished using the named semaphore indicated by *sem*. The effects of calling *sem_close*() for an unnamed semaphore (one created by *sem_init*()) are undefined. The *sem_close*() function shall deallocate (that is, make available for reuse by a subsequent *sem_open*() by this process) any system resources allocated by the system for use by this process for this semaphore. The effect of subsequent use of the semaphore indicated by *sem* by this process is undefined. If any threads in the calling process are currently blocked on the semaphore, the behavior is undefined. If the semaphore has not been removed with a successful call to *sem_unlink*(), then *sem_close*() has no effect on the state of the semaphore. If the *sem_unlink*() function has been successfully invoked for *name* after the most recent call to *sem_open*() with O_CREAT for this semaphore, then when all processes that have opened the semaphore close it, the semaphore is no longer accessible.

RETURN VALUE

Upon successful completion, a value of zero shall be returned. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sem_close*() function may fail if:

EINVAL
The *sem* argument is not a valid semaphore descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( ), *sem_unlink*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_destroy — destroy an unnamed semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_destroy(sem_t *sem);
```

DESCRIPTION

The *sem_destroy*() function shall destroy the unnamed semaphore indicated by *sem*. Only a semaphore that was created using *sem_init*() may be destroyed using *sem_destroy*(); the effect of calling *sem_destroy*() with a named semaphore is undefined. The effect of subsequent use of the semaphore *sem* is undefined until *sem* is reinitialized by another call to *sem_init*().

It is safe to destroy an initialized semaphore upon which no threads are currently blocked. The effect of destroying a semaphore upon which other threads are currently blocked is undefined.

RETURN VALUE

Upon successful completion, a value of zero shall be returned. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sem_destroy*() function may fail if:

EINVAL
The *sem* argument is not a valid semaphore.

EBUSY
There are currently processes blocked on the semaphore.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_init*( ), *sem_open*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_getvalue — get the value of a semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_getvalue(sem_t *restrict sem, int *restrict sval);
```

DESCRIPTION

The *sem_getvalue*() function shall update the location referenced by the *sval* argument to have the value of the semaphore referenced by *sem* without affecting the state of the semaphore. The updated value represents an actual semaphore value that occurred at some unspecified time during the call, but it need not be the actual value of the semaphore when it is returned to the calling process.

If *sem* is locked, then the object to which *sval* points shall either be set to zero or to a negative number whose absolute value represents the number of processes waiting for the semaphore at some unspecified time during the call.

RETURN VALUE

Upon successful completion, the *sem_getvalue*() function shall return a value of zero. Otherwise, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_getvalue*() function may fail if:

EINVAL
The *sem* argument does not refer to a valid semaphore.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_init — initialize an unnamed semaphore

SYNOPSIS

#include <semaphore.h>

int sem_init(sem_t **sem*, int *pshared*, unsigned *value*);

DESCRIPTION

The *sem_init*() function shall initialize the unnamed semaphore referred to by *sem*. The value of the initialized semaphore shall be *value*. Following a successful call to *sem_init*(), the semaphore may be used in subsequent calls to *sem_wait*(), *sem_timedwait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*(). This semaphore shall remain usable until the semaphore is destroyed.

If the *pshared* argument has a non-zero value, then the semaphore is shared between processes; in this case, any process that can access the semaphore *sem* can use *sem* for performing *sem_wait*(), *sem_timedwait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*() operations.

If the *pshared* argument is zero, then the semaphore is shared between threads of the process; any thread in this process can use *sem* for performing *sem_wait*(), *sem_timedwait*(), *sem_trywait*(), *sem_post*(), and *sem_destroy*() operations.

See *Section 2.9.9*, *Synchronization Object Copies and Alternative Mappings* for further requirements.

Attempting to initialize an already initialized semaphore results in undefined behavior.

RETURN VALUE

Upon successful completion, the *sem_init*() function shall initialize the semaphore in *sem* and return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *sem_init*() function shall fail if:

EINVAL

The *value* argument exceeds {SEM_VALUE_MAX}.

ENOSPC

A resource required to initialize the semaphore has been exhausted, or the limit on semaphores ({SEM_NSEMS_MAX}) has been reached.

EPERM

The process lacks appropriate privileges to initialize the semaphore.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_destroy( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_open — initialize and open a named semaphore

SYNOPSIS

```
#include <semaphore.h>

sem_t *sem_open(const char *name, int oflag, ...);
```

DESCRIPTION

The *sem_open*() function shall establish a connection between a named semaphore and a process. Following a call to *sem_open*() with semaphore name *name*, the process may reference the semaphore associated with *name* using the address returned from the call. This semaphore may be used in subsequent calls to *sem_wait*(), *sem_timedwait*(), *sem_trywait*(), *sem_post*(), and *sem_close*(). The semaphore remains usable by this process until the semaphore is closed by a successful call to *sem_close*(), *_exit*(), or one of the *exec* functions.

The *oflag* argument controls whether the semaphore is created or merely accessed by the call to *sem_open*(). The following flag bits may be set in *oflag*:

O_CREAT This flag is used to create a semaphore if it does not already exist. If O_CREAT is set and the semaphore already exists, then O_CREAT has no effect, except as noted under O_EXCL. Otherwise, *sem_open*() creates a named semaphore. The O_CREAT flag requires a third and a fourth argument: *mode*, which is of type **mode_t**, and *value*, which is of type **unsigned**. The semaphore is created with an initial value of *value*. Valid initial values for semaphores are less than or equal to {SEM_VALUE_MAX}.

The user ID of the semaphore shall be set to the effective user ID of the process. The group ID of the semaphore shall be set to the effective group ID of the process; however, if the *name* argument is visible in the file system, the group ID may be set to the group ID of the containing directory. The permission bits of the semaphore are set to the value of the *mode* argument except those set in the file mode creation mask of the process. When bits in *mode* other than the file permission bits are specified, the effect is unspecified.

After the semaphore named *name* has been created by *sem_open*() with the O_CREAT flag, other processes can connect to the semaphore by calling *sem_open*() with the same value of *name*.

O_EXCL If O_EXCL and O_CREAT are set, *sem_open*() fails if the semaphore *name* exists. The check for the existence of the semaphore and the creation of the semaphore if it does not exist are atomic with respect to other processes executing *sem_open*() with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the effect is undefined.

If flags other than O_CREAT and O_EXCL are specified in the *oflag* parameter, the effect is unspecified.

The *name* argument points to a string naming a semaphore object. It is unspecified whether the name appears in the file system and is visible to functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *sem_open*() with the same value of *name* shall refer to the same semaphore object, as long as that name has not been removed. If *name* does not begin with the <slash> character, the effect is implementation-defined.

If a process makes multiple successful calls to *sem_open*() with the same value for *name*, the same semaphore address shall be returned for each such successful call, provided that there have been no calls to *sem_unlink*() for this semaphore, and at least one previous successful *sem_open*() call for this semaphore has not been matched with a *sem_close*() call.

References to copies of the semaphore produce undefined results.

RETURN VALUE

Upon successful completion, the *sem_open*() function shall return the address of the semaphore. Otherwise, it shall return a value of SEM_FAILED and set *errno* to indicate the error. The symbol SEM_FAILED is defined in the <*semaphore.h*> header. No successful return from *sem_open*() shall return the value SEM_FAILED.

ERRORS

If any of the following conditions occur, the *sem_open*() function shall return SEM_FAILED and set *errno* to the corresponding value:

EACCES
The named semaphore exists and the permissions specified by *oflag* are denied, or the named semaphore does not exist and permission to create the named semaphore is denied.

EEXIST
O_CREAT and O_EXCL are set and the named semaphore already exists.

EINTR
The *sem_open*() operation was interrupted by a signal.

EINVAL
The *sem_open*() operation is not supported for the given name, or O_CREAT was specified in *oflag* and *value* was greater than {SEM_VALUE_MAX}.

EMFILE
Too many semaphore descriptors or file descriptors are currently in use by this process.

ENFILE
Too many semaphores are currently open in the system.

ENOENT
O_CREAT is not set and the named semaphore does not exist.

ENOMEM
There is insufficient memory for the creation of the new named semaphore.

ENOSPC
There is insufficient space on a storage device for the creation of the new named semaphore.

If any of the following conditions occur, the *sem_open*() function may return SEM_FAILED and set *errno* to the corresponding value:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Early drafts required an error return value of −1 with the type **sem_t *** for the *sem_open*() function, which is not guaranteed to be portable across implementations. The revised text provides the symbolic error code SEM_FAILED to eliminate the type conflict.

FUTURE DIRECTIONS

A future version might require the *sem_open*() and *sem_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_close*( ), *sem_post*( ), *sem_timedwait*( ), *sem_trywait*( ), *sem_unlink*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_post — unlock a semaphore

SYNOPSIS

#include <semaphore.h>

int sem_post(sem_t **sem*);

DESCRIPTION

The *sem_post*() function shall unlock the semaphore referenced by *sem* by performing a semaphore unlock operation on that semaphore.

If the semaphore value resulting from this operation is positive, then no threads were blocked waiting for the semaphore to become unlocked; the semaphore value is simply incremented.

If the value of the semaphore resulting from this operation is zero, then one of the threads blocked waiting for the semaphore shall be allowed to return successfully from its call to *sem_wait*(). If the Process Scheduling option is supported, the thread to be unblocked shall be chosen in a manner appropriate to the scheduling policies and parameters in effect for the blocked threads. In the case of the schedulers SCHED_FIFO and SCHED_RR, the highest priority waiting thread shall be unblocked, and if there is more than one highest priority thread blocked waiting for the semaphore, then the highest priority thread that has been waiting the longest shall be unblocked. If the Process Scheduling option is not defined, the choice of a thread to unblock is unspecified.

If the Process Sporadic Server option is supported, and the scheduling policy is SCHED_SPORADIC, the semantics are as per SCHED_FIFO above.

The *sem_post*() function shall be async-signal-safe and may be invoked from a signal-catching function.

RETURN VALUE

If successful, the *sem_post*() function shall return zero; otherwise, the function shall return −1 and set *errno* to indicate the error.

ERRORS

The *sem_post*() function may fail if:

EINVAL
The *sem* argument does not refer to a valid semaphore.

The following sections are informative.

EXAMPLES

See *sem_timedwait*( ).

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_timedwait*( ), *sem_trywait*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_timedwait — lock a semaphore

SYNOPSIS

```
#include <semaphore.h>
#include <time.h>

int sem_timedwait(sem_t *restrict sem,
   const struct timespec *restrict abstime);
```

DESCRIPTION

The *sem_timedwait*() function shall lock the semaphore referenced by *sem* as in the *sem_wait*() function. However, if the semaphore cannot be locked without waiting for another process or thread to unlock the semaphore by performing a *sem_post*() function, this wait shall be terminated when the specified timeout expires.

The timeout shall expire when the absolute time specified by *abstime* passes, as measured by the clock on which timeouts are based (that is, when the value of that clock equals or exceeds *abstime*), or if the absolute time specified by *abstime* has already been passed at the time of the call.

The timeout shall be based on the CLOCK_REALTIME clock. The resolution of the timeout shall be the resolution of the clock on which it is based. The **timespec** data type is defined as a structure in the *<time.h>* header.

Under no circumstance shall the function fail with a timeout if the semaphore can be locked immediately. The validity of the *abstime* need not be checked if the semaphore can be locked immediately.

RETURN VALUE

The *sem_timedwait*() function shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_timedwait*() function shall fail if:

EINVAL
The process or thread would have blocked, and the *abstime* parameter specified a nanoseconds field value less than zero or greater than or equal to 1 000 million.

ETIMEDOUT
The semaphore could not be locked before the specified timeout expired.

The *sem_timedwait*() function may fail if:

EDEADLK
A deadlock condition was detected.

EINTR
A signal interrupted this function.

EINVAL
The *sem* argument does not refer to a valid semaphore.

The following sections are informative.

EXAMPLES

The program shown below operates on an unnamed semaphore. The program expects two command-line arguments. The first argument specifies a seconds value that is used to set an alarm timer to generate a SIGALRM signal. This handler performs a *sem_post*(3) to increment the semaphore that is being waited on in *main*() using *sem_timedwait*(). The second command-line argument specifies the length of the timeout, in seconds, for *sem_timedwait*().

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <semaphore.h>
#include <time.h>
#include <assert.h>
#include <errno.h>
#include <signal.h>

sem_t sem;

static void
handler(int sig)
{
    int sav_errno = errno;
    static const char info_msg[] = "sem_post() from handler\n";
    write(STDOUT_FILENO, info_msg, sizeof info_msg - 1);
    if (sem_post(&sem) == -1) {
        static const char err_msg[] = "sem_post() failed\n";
        write(STDERR_FILENO, err_msg, sizeof err_msg - 1);
        _exit(EXIT_FAILURE);
    }
    errno = sav_errno;
}

int
main(int argc, char *argv[])
{
    struct sigaction sa;
    struct timespec ts;
    int s;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s <alarm-secs> <wait-secs>\n",
            argv[0]);
        exit(EXIT_FAILURE);
    }

    if (sem_init(&sem, 0, 0) == -1) {
        perror("sem_init");
        exit(EXIT_FAILURE);
    }

    /* Establish SIGALRM handler; set alarm timer using argv[1] */

    sa.sa_handler = handler;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    if (sigaction(SIGALRM, &sa, NULL) == -1) {
        perror("sigaction");
        exit(EXIT_FAILURE);
    }

    alarm(atoi(argv[1]));

    /* Calculate relative interval as current time plus
       number of seconds given argv[2] */
```

```
        if (clock_gettime(CLOCK_REALTIME, &ts) == -1) {
            perror("clock_gettime");
            exit(EXIT_FAILURE);
        }
        ts.tv_sec += atoi(argv[2]);

        printf("main() about to call sem_timedwait()\n");
        while ((s = sem_timedwait(&sem, &ts)) == -1 && errno == EINTR)
            continue;       /* Restart if interrupted by handler */

        /* Check what happened */

        if (s == -1) {
            if (errno == ETIMEDOUT)
                printf("sem_timedwait() timed out\n");
            else
                perror("sem_timedwait");
        } else
            printf("sem_timedwait() succeeded\n");

        exit((s == 0) ? EXIT_SUCCESS : EXIT_FAILURE);
    }
```

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

sem_post( ), *sem_trywait*( ), *semctl*( ), *semget*( ), *semop*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, **<semaphore.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_trywait, sem_wait — lock a semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_trywait(sem_t *sem);
int sem_wait(sem_t *sem);
```

DESCRIPTION

The *sem_trywait*() function shall lock the semaphore referenced by *sem* only if the semaphore is currently not locked; that is, if the semaphore value is currently positive. Otherwise, it shall not lock the semaphore.

The *sem_wait*() function shall lock the semaphore referenced by *sem* by performing a semaphore lock operation on that semaphore. If the semaphore value is currently zero, then the calling thread shall not return from the call to *sem_wait*() until it either locks the semaphore or the call is interrupted by a signal.

Upon successful return, the state of the semaphore shall be locked and shall remain locked until the *sem_post*() function is executed and returns successfully.

The *sem_wait*() function is interruptible by the delivery of a signal.

RETURN VALUE

The *sem_trywait*() and *sem_wait*() functions shall return zero if the calling process successfully performed the semaphore lock operation on the semaphore designated by *sem*. If the call was unsuccessful, the state of the semaphore shall be unchanged, and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_trywait*() function shall fail if:

EAGAIN
: The semaphore was already locked, so it cannot be immediately locked by the *sem_trywait*() operation.

The *sem_trywait*() and *sem_wait*() functions may fail if:

EDEADLK
: A deadlock condition was detected.

EINTR
: A signal interrupted this function.

EINVAL
: The *sem* argument does not refer to a valid semaphore.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions may be subject to priority inversion, as discussed in the Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_post*( ), *sem_timedwait*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.291*, *Priority Inversion*, *Section 4.12*, *Memory Synchronization*, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_unlink — remove a named semaphore

SYNOPSIS

#include <semaphore.h>

int sem_unlink(const char **name*);

DESCRIPTION

The *sem_unlink*() function shall remove the semaphore named by the string *name*. If the semaphore named by *name* is currently referenced by other processes, then *sem_unlink*() shall have no effect on the state of the semaphore. If one or more processes have the semaphore open when *sem_unlink*() is called, destruction of the semaphore is postponed until all references to the semaphore have been destroyed by calls to *sem_close*(), *_exit*(), or *exec*. Calls to *sem_open*() to recreate or reconnect to the semaphore refer to a new semaphore after *sem_unlink*() is called. The *sem_unlink*() call shall not block until all references have been destroyed; it shall return immediately.

RETURN VALUE

Upon successful completion, the *sem_unlink*() function shall return a value of 0. Otherwise, the semaphore shall not be changed and the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sem_unlink*() function shall fail if:

EACCES
Permission is denied to unlink the named semaphore.

ENOENT
The named semaphore does not exist.

The *sem_unlink*() function may fail if:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *sem_unlink*() with a *name* argument that contains the same semaphore name as was previously used in a successful *sem_open*() call shall not give an **[ENAMETOOLONG]** error.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

A future version might require the *sem_open*() and *sem_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

semctl( ), *semget*( ), *semop*( ), *sem_close*( ), *sem_open*( )

The Base Definitions volume of POSIX.1-2017, **<semaphore.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sem_wait — lock a semaphore

SYNOPSIS

```
#include <semaphore.h>

int sem_wait(sem_t *sem);
```

DESCRIPTION

Refer to *sem_trywait*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

semctl — XSI semaphore control operations

SYNOPSIS

```
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, ...);
```

DESCRIPTION

The *semctl*() function operates on XSI semaphores (see the Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *semctl*() function provides a variety of semaphore control operations as specified by *cmd*. The fourth argument is optional and depends upon the operation requested. If required, it is of type **union semun**, which the application shall explicitly declare:

```
union semun {
   int val;
   struct semid_ds *buf;
   unsigned short  *array;
} arg;
```

Each operation shall be performed atomically.

The following semaphore control operations as specified by *cmd* are executed with respect to the semaphore specified by *semid* and *semnum*. The level of permission required for each operation is shown with each command; see *Section 2.7*, *XSI Interprocess Communication*. The symbolic names for the values of *cmd* are defined in the *<sys/sem.h>* header:

GETVAL	Return the value of *semval*; see *<sys/sem.h>*. Requires read permission.
SETVAL	Set the value of *semval* to *arg.val*, where *arg* is the value of the fourth argument to *semctl*(). When this command is successfully executed, the *semadj* value corresponding to the specified semaphore in all processes is cleared. Also, the *sem_ctime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*. Requires alter permission; see *Section 2.7*, *XSI Interprocess Communication*.
GETPID	Return the value of *sempid*. Requires read permission.
GETNCNT	Return the value of *semncnt*. Requires read permission.
GETZCNT	Return the value of *semzcnt*. Requires read permission.

The following values of *cmd* operate on each *semval* in the set of semaphores:

GETALL	Return the value of *semval* for each semaphore in the semaphore set and place into the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*(). Requires read permission.
SETALL	Set the value of *semval* for each semaphore in the semaphore set according to the array pointed to by *arg.array*, where *arg* is the fourth argument to *semctl*(). When this command is successfully executed, the *semadj* values corresponding to each specified semaphore in all processes are cleared. Also, the *sem_ctime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*. Requires alter permission.

The following values of *cmd* are also available:

IPC_STAT Place the current value of each member of the **semid_ds** data structure associated with *semid* into the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*(). The contents of this structure are defined in <*sys/sem.h*>. Requires read permission.

IPC_SET Set the value of the following members of the **semid_ds** data structure associated with *semid* to the corresponding value found in the structure pointed to by *arg.buf*, where *arg* is the fourth argument to *semctl*():

sem_perm.uid
sem_perm.gid
sem_perm.mode

The mode bits specified in *Section 2.7.1*, *IPC General Description* are copied into the corresponding bits of the *sem_perm.mode* associated with *semid*. The stored values of any other bits are unspecified. The *sem_ctime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

IPC_RMID Remove the semaphore identifier specified by *semid* from the system and destroy the set of semaphores and **semid_ds** data structure associated with it. This command can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *sem_perm.cuid* or *sem_perm.uid* in the **semid_ds** data structure associated with *semid*.

RETURN VALUE

If successful, the value returned by *semctl*() depends on *cmd* as follows:

GETVAL	The value of *semval*.
GETPID	The value of *sempid*.
GETNCNT	The value of *semncnt*.
GETZCNT	The value of *semzcnt*.
All others	0.

Otherwise, *semctl*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *semctl*() function shall fail if:

EACCES
Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EINVAL
The value of *semid* is not a valid semaphore identifier, or the value of *semnum* is less than 0 or greater than or equal to *sem_nsems*, or the value of *cmd* is not a valid command.

EPERM
The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *sem_perm.cuid* or *sem_perm.uid* in the data structure associated with *semid*.

ERANGE
The argument *cmd* is equal to SETVAL or SETALL and the value to which *semval* is to be set is greater than the system-imposed maximum.

The following sections are informative.

EXAMPLES

Refer to *semop*( ).

APPLICATION USAGE

The fourth parameter in the SYNOPSIS section is now specified as **"..."** in order to avoid a clash with the ISO C standard when referring to the union *semun* (as defined in Issue 3) and for backwards-compatibility.

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *semget*( ), *semop*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*, **<sys_sem.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

semget — get set of XSI semaphores

SYNOPSIS

#include <sys/sem.h>

int semget(key_t *key*, int *nsems*, int *semflg*);

DESCRIPTION

The *semget*() function operates on XSI semaphores (see the Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *semget*() function shall return the semaphore identifier associated with *key*.

A semaphore identifier with its associated **semid_ds** data structure and its associated set of *nsems* semaphores (see *<sys/sem.h>*) is created for *key* if one of the following is true:

* The argument *key* is equal to IPC_PRIVATE.
* The argument *key* does not already have a semaphore identifier associated with it and (*semflg* &IPC_CREAT) is non-zero.

Upon creation, the **semid_ds** data structure associated with the new semaphore identifier is initialized as follows:

* In the operation permissions structure *sem_perm.cuid*, *sem_perm.uid*, *sem_perm.cgid*, and *sem_perm.gid* shall be set to the effective user ID and effective group ID, respectively, of the calling process.
* The low-order 9 bits of *sem_perm.mode* shall be set to the low-order 9 bits of *semflg*.
* The variable *sem_nsems* shall be set to the value of *nsems*.
* The variable *sem_otime* shall be set to 0 and *sem_ctime* shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.
* The data structure associated with each semaphore in the set need not be initialized. The *semctl*() function with the command SETVAL or SETALL can be used to initialize each semaphore.

RETURN VALUE

Upon successful completion, *semget*() shall return a non-negative integer, namely a semaphore identifier; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *semget*() function shall fail if:

EACCES
A semaphore identifier exists for *key*, but operation permission as specified by the low-order 9 bits of *semflg* would not be granted; see *Section 2.7*, *XSI Interprocess Communication*.

EEXIST
A semaphore identifier exists for the argument *key* but ((*semflg* &IPC_CREAT) &&(*semflg* &IPC_EXCL)) is non-zero.

EINVAL
The value of *nsems* is either less than or equal to 0 or greater than the system-imposed limit, or a semaphore identifier exists for the argument *key*, but the number of semaphores in the set associated with it is less than *nsems* and *nsems* is not equal to 0.

ENOENT
A semaphore identifier does not exist for the argument *key* and (*semflg* &IPC_CREAT) is equal to 0.

ENOSPC
A semaphore identifier is to be created but the system-imposed limit on the maximum number of allowed semaphores system-wide would be exceeded.

The following sections are informative.

EXAMPLES

Refer to *semop*( ).

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

A future version may require that the value of the *semval*, *sempid*, *semncnt*, and *semzcnt* members of all semaphores in a semaphore set be initialized to zero when a call to *semget*() creates a semaphore set. Many semaphore implementations already do this and it greatly simplifies what an application must do to initialize a semaphore set.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *ftok*( ), *semctl*( ), *semop*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*, **<sys_sem.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

semop — XSI semaphore operations

SYNOPSIS

#include <sys/sem.h>

int semop(int *semid*, struct sembuf **sops*, size_t *nsops*);

DESCRIPTION

The *semop*() function operates on XSI semaphores (see the Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *semop*() function shall perform atomically a user-defined array of semaphore operations in array order on the set of semaphores associated with the semaphore identifier specified by the argument *semid*.

The argument *sops* is a pointer to a user-defined array of semaphore operation structures. The implementation shall not modify elements of this array unless the application uses implementation-defined extensions.

The argument *nsops* is the number of such structures in the array.

Each structure, **sembuf**, includes the following members:

center box tab(!); cB | cB | cB lw(1.25i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ unsigned short!sem_num!Semaphore number. short!sem_op!Semaphore operation. short!sem_flg!Operation flags.

Each semaphore operation specified by *sem_op* is performed on the corresponding semaphore specified by *semid* and *sem_num*.

The variable *sem_op* specifies one of three semaphore operations:

1. If *sem_op* is a negative integer and the calling process has alter permission, one of the following shall occur:

   * If *semval*(see <*sys/sem.h*>) is greater than or equal to the absolute value of *sem_op*, the absolute value of *sem_op* is subtracted from *semval*. Also, if (*sem_flg* &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the *semadj* value of the calling process for the specified semaphore.

   * If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is non-zero, *semop*() shall return immediately.

   * If *semval* is less than the absolute value of *sem_op* and (*sem_flg* &IPC_NOWAIT) is 0, *semop*() shall increment the *semncnt* associated with the specified semaphore and suspend execution of the calling thread until one of the following conditions occurs:

     -- The value of *semval* becomes greater than or equal to the absolute value of *sem_op*. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented, the absolute value of *sem_op* shall be subtracted from *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the absolute value of *sem_op* shall be added to the *semadj* value of the calling process for the specified semaphore.

     -- The *semid* for which the calling thread is awaiting action is removed from the system. When this occurs, *errno* shall be set to **[EIDRM]** and −1 shall be returned.

     -- The calling thread receives a signal that is to be caught. When this occurs, the value of *semncnt* associated with the specified semaphore shall be decremented, and the calling thread shall resume execution in the manner prescribed in *sigaction*( ).

2. If *sem_op* is a positive integer and the calling process has alter permission, the value of *sem_op* shall be added to *semval* and, if (*sem_flg* &SEM_UNDO) is non-zero, the value of *sem_op* shall be subtracted from the *semadj* value of the calling process for the specified semaphore.
3. If *sem_op* is 0 and the calling process has read permission, one of the following shall occur:
   * If *semval* is 0, *semop*() shall return immediately.
   * If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is non-zero, *semop*() shall return immediately.
   * If *semval* is non-zero and (*sem_flg* &IPC_NOWAIT) is 0, *semop*() shall increment the *semzcnt* associated with the specified semaphore and suspend execution of the calling thread until one of the following occurs:
     -- The value of *semval* becomes 0, at which time the value of *semzcnt* associated with the specified semaphore shall be decremented.
     -- The *semid* for which the calling thread is awaiting action is removed from the system. When this occurs, *errno* shall be set to **[EIDRM]** and −1 shall be returned.
     -- The calling thread receives a signal that is to be caught. When this occurs, the value of *semzcnt* associated with the specified semaphore shall be decremented, and the calling thread shall resume execution in the manner prescribed in *sigaction*( ).

Upon successful completion, the value of *sempid* for each semaphore specified in the array pointed to by *sops* shall be set to the process ID of the calling process. Also, the *sem_otime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

RETURN VALUE

Upon successful completion, *semop*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *semop*() function shall fail if:

E2BIG The value of *nsops* is greater than the system-imposed maximum.

EACCES
Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EAGAIN
The operation would result in suspension of the calling process but (*sem_flg* &IPC_NOWAIT) is non-zero.

EFBIG
The value of *sem_num* is greater than or equal to the number of semaphores in the set associated with *semid*.

EIDRM
The semaphore identifier *semid* is removed from the system.

EINTR
The *semop*() function was interrupted by a signal.

EINVAL
The value of *semid* is not a valid semaphore identifier, or the number of individual semaphores for which the calling process requests a SEM_UNDO would exceed the system-imposed limit.

ENOSPC
The limit on the number of individual processes requesting a SEM_UNDO would be exceeded.

ERANGE
An operation would cause a *semval* to overflow the system-imposed limit, or an operation would cause a *semadj* value to overflow the system-imposed limit.

The following sections are informative.

EXAMPLES

Setting Values in Semaphores

The following example sets the values of the two semaphores associated with the *semid* identifier to the values contained in the *sb* array.

```
#include <sys/sem.h>
...
int semid;
struct sembuf sb[2];
int nsops = 2;
int result;

/* Code to initialize semid. */
...

/* Adjust value of semaphore in the semaphore array semid. */
sb[0].sem_num = 0;
sb[0].sem_op = -1;
sb[0].sem_flg = SEM_UNDO | IPC_NOWAIT;
sb[1].sem_num = 1;
sb[1].sem_op = 1;
sb[1].sem_flg = 0;

result = semop(semid, sb, nsops);
```

Creating a Semaphore Identifier

The following example gets a unique semaphore key using the *ftok*() function, then gets a semaphore ID associated with that key using the *semget*() function (the first call also tests to make sure the semaphore exists). If the semaphore does not exist, the program creates it, as shown by the second call to *semget*(). In creating the semaphore for the queuing process, the program attempts to create one semaphore with read/write permission for all. It also uses the IPC_EXCL flag, which forces *semget*() to fail if the semaphore already exists.

After creating the semaphore, the program uses calls to *semctl*() and *semop*() to initialize it to the values in the *sbuf* array. The number of processes that can execute concurrently without queuing is initially set to 2. The final call to *semget*() creates a semaphore identifier that can be used later in the program.

Processes that obtain *semid* without creating it check that *sem_otime* is non-zero, to ensure that the creating process has completed the *semop*() initialization.

The final call to *semop*() acquires the semaphore and waits until it is free; the SEM_UNDO option releases the semaphore when the process exits, waiting until there are less than two processes running concurrently.

```
#include <stdio.h>
#include <sys/sem.h>
#include <sys/stat.h>
#include <errno.h>
#include <stdlib.h>
...
key_t semkey;
int semid;
struct sembuf sbuf;
union semun {
    int val;
    struct semid_ds *buf;
    unsigned short *array;
} arg;
struct semid_ds ds;
...
/* Get unique key for semaphore. */
if ((semkey = ftok("/tmp", 'a')) == (key_t) -1) {
    perror("IPC error: ftok"); exit(1);
}

/* Get semaphore ID associated with this key. */
if ((semid = semget(semkey, 0, 0)) == -1) {

    /* Semaphore does not exist - Create. */
    if ((semid = semget(semkey, 1, IPC_CREAT | IPC_EXCL | S_IRUSR |
        S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH)) != -1)
    {
        /* Initialize the semaphore. */
        arg.val = 0;
        sbuf.sem_num = 0;
        sbuf.sem_op = 2;  /* This is the number of runs without queuing. */
        sbuf.sem_flg = 0;
        if (semctl(semid, 0, SETVAL, arg) == -1
            || semop(semid, &sbuf, 1) == -1) {
            perror("IPC error: semop"); exit(1);
        }
    }
    else if (errno == EEXIST) {
        if ((semid = semget(semkey, 0, 0)) == -1) {
            perror("IPC error 1: semget"); exit(1);
        }
        goto check_init;
    }
    else {
        perror("IPC error 2: semget"); exit(1);
    }
}
else
{
    /* Check that semid has completed initialization. */
    /* An application can use a retry loop at this point rather than
       exiting. */
    check_init:
    arg.buf = &ds;
    if (semctl(semid, 0, IPC_STAT, arg) < 0) {
        perror("IPC error 3: semctl"); exit(1);
    }
    if (ds.sem_otime == 0) {
        perror("IPC error 4: semctl"); exit(1);
    }
}
...
sbuf.sem_num = 0;
sbuf.sem_op = -1;
sbuf.sem_flg = SEM_UNDO;
if (semop(semid, &sbuf, 1) == -1) {
    perror("IPC Error: semop"); exit(1);
    }
```

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *exec*, *exit*( ), *fork*( ), *semctl*( ), *semget*( ), *sem_close*( ), *sem_destroy*( ), *sem_getvalue*( ), *sem_init*( ), *sem_open*( ), *sem_post*( ), *sem_trywait*( ), *sem_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.17*, *Semaphore*, **<sys_ipc.h>**, **<sys_sem.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

send — send a message on a socket

SYNOPSIS

#include <sys/socket.h>

ssize_t send(int *socket*, const void **buffer*, size_t *length*, int *flags*);

DESCRIPTION

The *send*() function shall initiate transmission of a message from the specified socket to its peer. The *send*() function shall send a message only when the socket is connected. If the socket is a connectionless-mode socket, the message shall be sent to the pre-specified peer address.

The *send*() function takes the following arguments:

socket	Specifies the socket file descriptor.
buffer	Points to the buffer containing the message to send.
length	Specifies the length of the message in bytes.
flags	Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:

MSG_EOR	Terminates a record (if supported by the protocol).
MSG_OOB	Sends out-of-band data on sockets that support out-of-band communications. The significance and semantics of out-of-band data are protocol-specific.
MSG_NOSIGNAL	Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The **[EPIPE]** error shall still be returned.

The length of the message to be sent is specified by the *length* argument. If the message is too long to pass through the underlying protocol, *send*() shall fail and no data shall be transmitted.

Successful completion of a call to *send*() does not guarantee delivery of the message. A return value of −1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does not have O_NONBLOCK set, *send*() shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted, and the socket file descriptor does have O_NONBLOCK set, *send*() shall fail. The *select*() and *poll*() functions can be used to determine when it is possible to send more data.

The socket in use may require the process to have appropriate privileges to use the *send*() function.

RETURN VALUE

Upon successful completion, *send*() shall return the number of bytes sent. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *send*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EDESTADDRREQ
The socket is not connection-mode and no peer address is set.

EINTR
A signal interrupted *send*() before any data was transmitted.

EMSGSIZE
The message is too large to be sent all at once, as the socket requires.

ENOTCONN
The socket is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

EPIPE The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.

The *send*() function may fail if:

EACCES
The calling process does not have appropriate privileges.

EIO An I/O error occurred while reading from or writing to the file system.

ENETDOWN
The local network interface used to reach the destination is down.

ENETUNREACH
No route to the network is present.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the *socket* argument refers to a connection-mode socket, the *send*() function is equivalent to *sendto*() (with any value for the *dest_addr* and *dest_len* arguments, as they are ignored in this case). If the *socket* argument refers to a socket and the *flags* argument is 0, the *send*() function is equivalent to *write*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

connect( ), *getsockopt*( ), *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *sendmsg*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sendmsg — send a message on a socket using a message structure

SYNOPSIS

#include <sys/socket.h>

ssize_t sendmsg(int *socket*, const struct msghdr **message*, int *flags*);

DESCRIPTION

The *sendmsg*() function shall send a message through a connection-mode or connectionless-mode socket. If the socket is a connectionless-mode socket, the message shall be sent to the address specified by **msghdr** if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified in **msghdr** (overriding the pre-specified peer address), or the function shall return −1 and set *errno* to **[EISCONN]**. If the socket is connection-mode, the destination address in **msghdr** shall be ignored.

The *sendmsg*() function takes the following arguments:

socket Specifies the socket file descriptor.

message Points to a **msghdr** structure, containing both the destination address and the buffers for the outgoing message. The length and format of the address depend on the address family of the socket. The *msg_flags* member is ignored.

flags Specifies the type of message transmission. The application may specify 0 or the following flag:

MSG_EOR Terminates a record (if supported by the protocol).

MSG_OOB Sends out-of-band data on sockets that support out-of-bound data. The significance and semantics of out-of-band data are protocol-specific.

MSG_NOSIGNAL
Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The **[EPIPE]** error shall still be returned.

The *msg_iov* and *msg_iovlen* fields of *message* specify zero or more buffers containing the data to be sent. *msg_iov* points to an array of **iovec** structures; *msg_iovlen* shall be set to the dimension of this array. In each **iovec** structure, the *iov_base* field specifies a storage area and the *iov_len* field gives its size in bytes. Some of these sizes can be zero. The data from each storage area indicated by *msg_iov* is sent in turn.

Successful completion of a call to *sendmsg*() does not guarantee delivery of the message. A return value of −1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, the *sendmsg*() function shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, the *sendmsg*() function shall fail.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendmsg*() shall fail if the SO_BROADCAST option is not set for the socket.

The socket in use may require the process to have appropriate privileges to use the *sendmsg*() function.

RETURN VALUE

Upon successful completion, *sendmsg*() shall return the number of bytes sent. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sendmsg*() function shall fail if:

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.

EAFNOSUPPORT
Addresses in the specified address family cannot be used with this socket.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EINTR
A signal interrupted *sendmsg*() before any data was transmitted.

EINVAL
The sum of the *iov_len* values overflows an **ssize_t**.

EMSGSIZE
The message is too large to be sent all at once (as the socket requires), or the *msg_iovlen* member of the **msghdr** structure pointed to by *message* is less than or equal to 0 or is greater than {IOV_MAX}.

ENOTCONN
The socket is connection-mode but is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

EPIPE The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.

If the address family of the socket is AF_UNIX, then *sendmsg*() shall fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the pathname in the socket address.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the pathname does not name an existing file or the path name is an empty string.

ENOTDIR
A component of the path prefix of the pathname in the socket address names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in the socket address contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *sendmsg*() function may fail if:

EACCES
Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

EDESTADDRREQ
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

EHOSTUNREACH
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

EIO An I/O error occurred while reading from or writing to the file system.

EISCONN
A destination address was specified and the socket is already connected.

ENETDOWN
The local network interface used to reach the destination is down.

ENETUNREACH
No route to the network is present.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

If the address family of the socket is AF_UNIX, then *sendmsg*() may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

Done.

APPLICATION USAGE

The *select*() and *poll*() functions can be used to determine when it is possible to send more data.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendto*( ), *setsockopt*( ), *shutdown*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sendto — send a message on a socket

SYNOPSIS

```
#include <sys/socket.h>

ssize_t sendto(int socket, const void *message, size_t length,
   int flags, const struct sockaddr *dest_addr,
   socklen_t dest_len);
```

DESCRIPTION

The *sendto*() function shall send a message through a connection-mode or connectionless-mode socket.

If the socket is a connectionless-mode socket, the message shall be sent to the address specified by *dest_addr* if no pre-specified peer address has been set. If a peer address has been pre-specified, either the message shall be sent to the address specified by *dest_addr* (overriding the pre-specified peer address), or the function shall return −1 and set *errno* to **[EISCONN]**.

If the socket is connection-mode, *dest_addr* shall be ignored.

The *sendto*() function takes the following arguments:

socket Specifies the socket file descriptor.

message Points to a buffer containing the message to be sent.

length Specifies the size of the message in bytes.

flags Specifies the type of message transmission. Values of this argument are formed by logically OR'ing zero or more of the following flags:

MSG_EOR Terminates a record (if supported by the protocol).

MSG_OOB Sends out-of-band data on sockets that support out-of-band data. The significance and semantics of out-of-band data are protocol-specific.

MSG_NOSIGNAL
Requests not to send the SIGPIPE signal if an attempt to send is made on a stream-oriented socket that is no longer connected. The **[EPIPE]** error shall still be returned.

dest_addr Points to a **sockaddr** structure containing the destination address. The length and format of the address depend on the address family of the socket.

dest_len Specifies the length of the **sockaddr** structure pointed to by the *dest_addr* argument.

If the socket protocol supports broadcast and the specified address is a broadcast address for the socket protocol, *sendto*() shall fail if the SO_BROADCAST option is not set for the socket.

The *dest_addr* argument specifies the address of the target.

The *length* argument specifies the length of the message.

Successful completion of a call to *sendto*() does not guarantee delivery of the message. A return value of −1 indicates only locally-detected errors.

If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does not have O_NONBLOCK set, *sendto*() shall block until space is available. If space is not available at the sending socket to hold the message to be transmitted and the socket file descriptor does have O_NONBLOCK set, *sendto*() shall fail.

The socket in use may require the process to have appropriate privileges to use the *sendto*() function.

RETURN VALUE

Upon successful completion, *sendto*() shall return the number of bytes sent. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sendto*() function shall fail if:

EAFNOSUPPORT
Addresses in the specified address family cannot be used with this socket.

EAGAIN or **EWOULDBLOCK**
The socket's file descriptor is marked O_NONBLOCK and the requested operation would block.

EBADF
The *socket* argument is not a valid file descriptor.

ECONNRESET
A connection was forcibly closed by a peer.

EINTR
A signal interrupted *sendto*() before any data was transmitted.

EMSGSIZE
The message is too large to be sent all at once, as the socket requires.

ENOTCONN
The socket is connection-mode but is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

EOPNOTSUPP
The *socket* argument is associated with a socket that does not support one or more of the values set in *flags*.

EPIPE The socket is shut down for writing, or the socket is connection-mode and is no longer connected. In the latter case, and if the socket is of type SOCK_STREAM or SOCK_SEQPACKET and the MSG_NOSIGNAL flag is not set, the SIGPIPE signal is generated to the calling thread.

If the address family of the socket is AF_UNIX, then *sendto*() shall fail if:

EIO An I/O error occurred while reading from or writing to the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the pathname in the socket address.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of the pathname does not name an existing file or the pathname is an empty string.

ENOTDIR
A component of the path prefix of the pathname in the socket address names an existing file that is neither a directory nor a symbolic link to a directory, or the pathname in the socket address contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

The *sendto*() function may fail if:

EACCES
Search permission is denied for a component of the path prefix; or write access to the named socket is denied.

EDESTADDRREQ
The socket is not connection-mode and does not have its peer address set, and no destination address was specified.

EHOSTUNREACH
The destination host cannot be reached (probably because the host is down or a remote router cannot reach it).

EINVAL
The *dest_len* argument is not a valid length for the address family.

EIO An I/O error occurred while reading from or writing to the file system.

EISCONN
A destination address was specified and the socket is already connected.

ENETDOWN
The local network interface used to reach the destination is down.

ENETUNREACH
No route to the network is present.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

If the address family of the socket is AF_UNIX, then *sendto*() may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the pathname in the socket address.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
The *select*() and *poll*() functions can be used to determine when it is possible to send more data.

RATIONALE
None.

FUTURE DIRECTIONS
None.

SEE ALSO
getsockopt( ), *poll*( ), *pselect*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *setsockopt*( ), *shutdown*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT
Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setbuf — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

void setbuf(FILE *restrict stream, char *restrict buf);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

Except that it returns no value, the function call:

```
setbuf(stream, buf)
```

shall be equivalent to:

```
setvbuf(stream, buf, _IOFBF, BUFSIZ)
```

if *buf* is not a null pointer, or to:

```
setvbuf(stream, buf, _IONBF, BUFSIZ)
```

if *buf* is a null pointer.

RETURN VALUE

The *setbuf*() function shall not return a value.

ERRORS

Although the *setvbuf*() interface may set *errno* in defined ways, the value of *errno* after a call to *setbuf*() is unspecified.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

With *setbuf*(), allocating a buffer of BUFSIZ bytes does not necessarily imply that all of BUFSIZ bytes are used for the buffer area.

Since *errno* is not required to be unchanged on success, in order to correctly detect and possibly recover from errors, applications should use *setvbuf*() instead of *setbuf*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *setvbuf*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setegid — set the effective group ID

SYNOPSIS

```
#include <unistd.h>

int setegid(gid_t gid);
```

DESCRIPTION

If *gid* is equal to the real group ID or the saved set-group-ID, or if the process has appropriate privileges, *setegid*() shall set the effective group ID of the calling process to *gid*; the real group ID, saved set-group-ID, and any supplementary group IDs shall remain unchanged.

The *setegid*() function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setegid*() function shall fail if:

EINVAL
The value of the *gid* argument is invalid and is not supported by the implementation.

EPERM
The process does not have appropriate privileges and *gid* does not match the real group ID or the saved set-group-ID.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setenv — add or change environment variable

SYNOPSIS

#include <stdlib.h>

int setenv(const char **envname*, const char **envval*, int *overwrite*);

DESCRIPTION

The *setenv*() function shall update or add a variable in the environment of the calling process. The *envname* argument points to a string containing the name of an environment variable to be added or altered. The environment variable shall be set to the value to which *envval* points. The function shall fail if *envname* points to a string which contains an **'='** character. If the environment variable named by *envname* already exists and the value of *overwrite* is non-zero, the function shall return success and the environment shall be updated. If the environment variable named by *envname* already exists and the value of *overwrite* is zero, the function shall return success and the environment shall remain unchanged.

The *setenv*() function shall update the list of pointers to which *environ* points.

The strings described by *envname* and *envval* are copied by this function.

The *setenv*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, −1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *setenv*() function shall fail if:

EINVAL

The *envname* argument points to an empty string or points to a string containing an **'='** character.

ENOMEM

Insufficient memory was available to add a variable or its value to the environment.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

See *exec*() for restrictions on changing the environment in multi-threaded applications.

RATIONALE

Unanticipated results may occur if *setenv*() changes the external variable *environ*. In particular, if the optional *envp* argument to *main*() is present, it is not changed, and thus may point to an obsolete copy of the environment (as may any other copy of *environ*). However, other than the aforementioned restriction, the standard developers intended that the traditional method of walking through the environment by way of the *environ* pointer must be supported.

It was decided that *setenv*() should be required by this version because it addresses a piece of missing functionality, and does not impose a significant burden on the implementor.

There was considerable debate as to whether the System V *putenv*() function or the BSD *setenv*() function should be required as a mandatory function. The *setenv*() function was chosen because it permitted the implementation of the *unsetenv*() function to delete environmental variables, without specifying an additional interface. The *putenv*() function is available as part of the XSI option.

The standard developers considered requiring that *setenv*() indicate an error when a call to it would result in exceeding {ARG_MAX}. The requirement was rejected since the condition might be temporary, with the application eventually reducing the environment size. The ultimate success or failure depends on the size at the time of a call to *exec*, which returns an indication of this error condition.

See also the RATIONALE section in *getenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getenv*( ), *putenv*( ), *unsetenv*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

seteuid — set effective user ID

SYNOPSIS

```
#include <unistd.h>

int seteuid(uid_t uid);
```

DESCRIPTION

If *uid* is equal to the real user ID or the saved set-user-ID, or if the process has appropriate privileges, *seteuid*() shall set the effective user ID of the calling process to *uid*; the real user ID and saved set-user-ID shall remain unchanged.

The *seteuid*() function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *seteuid*() function shall fail if:

EINVAL
The value of the *uid* argument is invalid and is not supported by the implementation.

EPERM
The process does not have appropriate privileges and *uid* does not match the real user ID or the saved set-user-ID.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *setgid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setgid — set-group-ID

SYNOPSIS

```
#include <unistd.h>

int setgid(gid_t gid);
```

DESCRIPTION

If the process has appropriate privileges, *setgid*() shall set the real group ID, effective group ID, and the saved set-group-ID of the calling process to *gid*.

If the process does not have appropriate privileges, but *gid* is equal to the real group ID or the saved set-group-ID, *setgid*() shall set the effective group ID to *gid*; the real group ID and saved set-group-ID shall remain unchanged.

The *setgid*() function shall not affect the supplementary group list in any way.

Any supplementary group IDs of the calling process shall remain unchanged.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setgid*() function shall fail if:

EINVAL
The value of the *gid* argument is invalid and is not supported by the implementation.

EPERM
The process does not have appropriate privileges and *gid* does not match the real group ID or the saved set-group-ID.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setuid*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setregid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setgrent — reset the group database to the first entry

SYNOPSIS

```
#include <grp.h>

void setgrent(void);
```

DESCRIPTION

Refer to *endgrent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sethostent — network host database functions

SYNOPSIS

```
#include <netdb.h>

void sethostent(int stayopen);
```

DESCRIPTION

Refer to *endhostent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setitimer — set the value of an interval timer

SYNOPSIS

```
#include <sys/time.h>

int setitimer(int which, const struct itimerval *restrict value,
    struct itimerval *restrict ovalue);
```

DESCRIPTION

Refer to *getitimer*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setjmp — set jump point for a non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(jmp_buf env);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

A call to *setjmp*() shall save the calling environment in its *env* argument for later use by *longjmp*().

It is unspecified whether *setjmp*() is a macro or a function. If a macro definition is suppressed in order to access an actual function, or a program defines an external identifier with the name *setjmp*, the behavior is undefined.

An application shall ensure that an invocation of *setjmp*() appears in one of the following contexts only:

* The entire controlling expression of a selection or iteration statement
* One operand of a relational or equality operator with the other operand an integral constant expression, with the resulting expression being the entire controlling expression of a selection or iteration statement
* The operand of a unary **'!'** operator with the resulting expression being the entire controlling expression of a selection or iteration
* The entire expression of an expression statement (possibly cast to **void**)

If the invocation appears in any other context, the behavior is undefined.

RETURN VALUE

If the return is from a direct invocation, *setjmp*() shall return 0. If the return is from a call to *longjmp*(), *setjmp*() shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

In general, *sigsetjmp*() is more useful in dealing with errors and interrupts encountered in a low-level subroutine of a program.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp( ), *sigsetjmp*( )

The Base Definitions volume of POSIX.1-2017, **<setjmp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setkey — set encoding key (**CRYPT**)

SYNOPSIS

#include <stdlib.h>

void setkey(const char **key*);

DESCRIPTION

The *setkey*() function provides access to an implementation-defined encoding algorithm. The argument of *setkey*() is an array of length 64 bytes containing only the bytes with numerical value of 0 and 1. If this string is divided into groups of 8, the low-order bit in each group is ignored; this gives a 56-bit key which is used by the algorithm. This is the key that shall be used with the algorithm to encode a string *block* passed to *encrypt*().

The *setkey*() function shall not change the setting of *errno* if successful. An application wishing to check for error situations should set *errno* to 0 before calling *setkey*(). If *errno* is non-zero on return, an error has occurred.

The *setkey*() function need not be thread-safe.

RETURN VALUE

No values are returned.

ERRORS

The *setkey*() function shall fail if:

ENOSYS
The functionality is not supported on this implementation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Decoding need not be implemented in all environments. This is related to government restrictions in some countries on encryption and decryption routines. Historical practice has been to ship a different version of the encryption library without the decryption feature in the routines supplied. Thus the exported version of *encrypt*() does encoding but not decoding.

RATIONALE

None.

FUTURE DIRECTIONS

A future version of the standard may mark this interface as obsolete or remove it altogether.

SEE ALSO

crypt( ), *encrypt*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setlocale — set program locale

SYNOPSIS

#include <locale.h>

char *setlocale(int *category*, const char **locale*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *setlocale*() function selects the appropriate piece of the global locale, as specified by the *category* and *locale* arguments, and can be used to change or query the entire global locale or portions thereof. The value LC_ALL for *category* names the entire global locale; other values for *category* name only a part of the global locale:

LC_COLLATE
: Affects the behavior of regular expressions and the collation functions.

LC_CTYPE
: Affects the behavior of regular expressions, character classification, character conversion functions, and wide-character functions.

LC_MESSAGES
: Affects the affirmative and negative response expressions returned by *nl_langinfo*() and the way message catalogs are located. It may also affect the behavior of functions that return or write message strings.

LC_MONETARY
: Affects the behavior of functions that handle monetary values.

LC_NUMERIC
: Affects the behavior of functions that handle numeric values.

LC_TIME
: Affects the behavior of the time conversion functions.

The *locale* argument is a pointer to a character string containing the required setting of *category*. The contents of this string are implementation-defined. In addition, the following preset values of *locale* are defined for all settings of *category*:

"POSIX"
: Specifies the minimal environment for C-language translation called the POSIX locale. The POSIX locale is the default global locale at entry to *main*().

"C"
: Equivalent to **"POSIX"**.

" "
: Specifies an implementation-defined native environment. The determination of the name of the new locale for the specified category depends on the value of the associated environment variables, *LC_** and *LANG*; see the Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale* and *Chapter 8*, *Environment Variables*.

A null pointer
: Directs *setlocale*() to query the current global locale setting and return the name of the locale if *category* is not LC_ALL, or a string which encodes the locale name(s) for all of the individual categories if *category* is LC_ALL.

Setting all of the categories of the global locale is similar to successively setting each individual category of the global locale, except that all error checking is done before any actions are performed. To set all the categories of the global locale, *setlocale*() can be invoked as:

```
    setlocale(LC_ALL, "");
```

In this case, *setlocale*() shall first verify that the values of all the environment variables it needs according to the precedence rules (described in the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*) indicate supported locales. If the value of any of these environment variable searches yields a locale that is not supported (and non-null), *setlocale*() shall return a null pointer and the global locale shall not be changed. If all environment variables name supported locales, *setlocale*() shall proceed as if it had been called for each category, using the appropriate value from the associated environment variable or from the implementation-defined default if there is no such value.

The global locale established using *setlocale*() shall only be used in threads for which no current locale has been set using *uselocale*() or whose current locale has been set to the global locale using *uselocale*(LC_GLOBAL_LOCALE).

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *setlocale*().

The *setlocale*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *setlocale*() shall return the string associated with the specified category for the new locale. Otherwise, *setlocale*() shall return a null pointer and the global locale shall not be changed.

A null pointer for *locale* shall cause *setlocale*() to return a pointer to the string associated with the specified *category* for the current global locale. The global locale shall not be changed.

The string returned by *setlocale*() is such that a subsequent call with that string and its associated *category* shall restore that part of the global locale. The application shall not modify the string returned. The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to *setlocale*(). The returned pointer might also be invalidated if the calling thread is terminated.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The following code illustrates how a program can initialize the international environment for one language, while selectively modifying the global locale such that regular expressions and string operations can be applied to text recorded in a different language:

```
    setlocale(LC_ALL, "De");
    setlocale(LC_COLLATE, "Fr@dict");
```

Internationalized programs can initiate language operation according to environment variable settings (see the Base Definitions volume of POSIX.1-2017, *Section 8.2*, *Internationalization Variables*) by calling *setlocale*() as follows:

```
    setlocale(LC_ALL, "");
```

Changing the setting of *LC_MESSAGES* has no effect on catalogs that have already been opened by calls to *catopen*().

In order to make use of different locale settings while multiple threads are running, applications should use *uselocale*() in preference to *setlocale*().

RATIONALE

References to the international environment or locale in the following text relate to the global locale for the process. This can be overridden for individual threads using *uselocale*().

The ISO C standard defines a collection of functions to support internationalization. One of the most significant aspects of these functions is a facility to set and query the *international environment*. The international environment is a repository of information that affects the behavior of certain functionality, namely:

1. Character handling
2. Collating
3. Date/time formatting
4. Numeric editing
5. Monetary formatting
6. Messaging

The *setlocale*() function provides the application developer with the ability to set all or portions, called *categories*, of the international environment. These categories correspond to the areas of functionality mentioned above. The syntax for *setlocale*() is as follows:

```
char *setlocale(int category, const char *locale);
```

where *category* is the name of one of following categories, namely:

LC_COLLATE LC_CTYPE LC_MESSAGES LC_MONETARY LC_NUMERIC LC_TIME

In addition, a special value called LC_ALL directs *setlocale*() to set all categories.

There are two primary uses of *setlocale*():

1. Querying the international environment to find out what it is set to
2. Setting the international environment, or *locale*, to a specific value

The behavior of *setlocale*() in these two areas is described below. Since it is difficult to describe the behavior in words, examples are used to illustrate the behavior of specific uses.

To query the international environment, *setlocale*() is invoked with a specific category and the null pointer as the locale. The null pointer is a special directive to *setlocale*() that tells it to query rather than set the international environment. The following syntax is used to query the name of the international environment:

```
setlocale({LC_ALL, LC_COLLATE, LC_CTYPE, LC_MESSAGES, LC_MONETARY, \
  LC_NUMERIC, LC_TIME},(char *) NULL);
```

The *setlocale*() function shall return the string corresponding to the current international environment. This value may be used by a subsequent call to *setlocale*() to reset the international environment to this value. However, it should be noted that the return value from *setlocale*() may be a pointer to a static area within the function and is not guaranteed to remain unchanged (that is, it may be modified by a subsequent call to *setlocale*()). Therefore, if the purpose of calling *setlocale*() is to save the value of the current international environment so it can be changed and reset later, the return value should be copied to an array of **char** in the calling program.

There are three ways to set the international environment with *setlocale*():

setlocale(*category*, *string*)
: This usage sets a specific *category* in the international environment to a specific value corresponding to the value of the *string*. A specific example is provided below:

```
setlocale(LC_ALL, "fr_FR.ISO-8859-1");
```

In this example, all categories of the international environment are set to the locale corresponding to the string **"fr_FR.ISO-8859-1"**, or to the French language as spoken in France using the ISO/IEC 8859-1: 1998 standard codeset.

If the string does not correspond to a valid locale, *setlocale*() shall return a null pointer and the international environment is not changed. Otherwise, *setlocale*() shall return the name of the locale just set.

setlocale(*category*, "C")
The ISO C standard states that one locale must exist on all conforming implementations. The name of the locale is C and corresponds to a minimal international environment needed to support the C programming language.

setlocale(*category*, "")
This sets a specific category to an implementation-defined default. This corresponds to the value of the environment variables.

FUTURE DIRECTIONS

None.

SEE ALSO

catopen( ), *exec*, *fprintf*( ), *fscanf*( ), *isalnum*( ), *isalpha*( ), *isblank*( ), *iscntrl*( ), *isdigit*( ), *isgraph*( ), *islower*( ), *isprint*( ), *ispunct*( ), *isspace*( ), *isupper*( ), *iswalnum*( ), *iswalpha*( ), *iswblank*( ), *iswcntrl*( ), *iswctype*( ), *iswdigit*( ), *iswgraph*( ), *iswlower*( ), *iswprint*( ), *iswpunct*( ), *iswspace*( ), *iswupper*( ), *iswxdigit*( ), *isxdigit*( ), *localeconv*( ), *mblen*( ), *mbstowcs*( ), *mbtowc*( ), *newlocale*( ), *nl_langinfo*( ), *perror*( ), *psiginfo*( ), *strcoll*( ), *strerror*( ), *strfmon*( ), *strsignal*( ), *strtod*( ), *strxfrm*( ), *tolower*( ), *toupper*( ), *towlower*( ), *towupper*( ), *uselocale*( ), *wcscoll*( ), *wcstod*( ), *wcstombs*( ), *wcsxfrm*( ), *wctomb*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, *Chapter 8*, *Environment Variables*, **<langinfo.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setlogmask — set the log priority mask

SYNOPSIS

#include <syslog.h>

int setlogmask(int *maskpri*);

DESCRIPTION

Refer to *closelog*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setnetent — network database function

SYNOPSIS

#include <netdb.h>

void setnetent(int *stayopen*);

DESCRIPTION

Refer to *endnetent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setpgid — set process group ID for job control

SYNOPSIS

```
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

DESCRIPTION

The *setpgid*() function shall either join an existing process group or create a new process group within the session of the calling process.

The process group ID of a session leader shall not change.

Upon successful completion, the process group ID of the process with a process ID that matches *pid* shall be set to *pgid*.

As a special case, if *pid* is 0, the process ID of the calling process shall be used. Also, if *pgid* is 0, the process ID of the indicated process shall be used.

RETURN VALUE

Upon successful completion, *setpgid*() shall return 0; otherwise, −1 shall be returned and *errno* shall be set to indicate the error.

ERRORS

The *setpgid*() function shall fail if:

EACCES
The value of the *pid* argument matches the process ID of a child process of the calling process and the child process has successfully executed one of the *exec* functions.

EINVAL
The value of the *pgid* argument is less than 0, or is not a value supported by the implementation.

EPERM
The process indicated by the *pid* argument is a session leader.

EPERM
The value of the *pid* argument matches the process ID of a child process of the calling process and the child process is not in the same session as the calling process.

EPERM
The value of the *pgid* argument is valid but does not match the process ID of the process indicated by the *pid* argument and there is no process with a process group ID that matches the value of the *pgid* argument in the same session as the calling process.

ESRCH
The value of the *pid* argument does not match the process ID of the calling process or of a child process of the calling process.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setpgid*() function shall group processes together for the purpose of signaling, placement in foreground or background, and other job control actions.

The *setpgid*() function is similar to the *setpgrp*() function of 4.2 BSD, except that 4.2 BSD allowed the specified new process group to assume any value. This presents certain security problems and is more flexible than necessary to support job control.

To provide tighter security, *setpgid*() only allows the calling process to join a process group already in use inside its session or create a new process group whose process group ID was equal to its process ID.

When a job control shell spawns a new job, the processes in the job must be placed into a new process group via *setpgid*(). There are two timing constraints involved in this action:

1. The new process must be placed in the new process group before the appropriate program is launched via one of the *exec* functions.

2. The new process must be placed in the new process group before the shell can correctly send signals to the new process group.

To address these constraints, the following actions are performed. The new processes call *setpgid*() to alter their own process groups after *fork*() but before *exec*. This satisfies the first constraint. Under 4.3 BSD, the second constraint is satisfied by the synchronization property of *vfork*( ); that is, the shell is suspended until the child has completed the *exec*, thus ensuring that the child has completed the *setpgid*(). A new version of *fork*() with this same synchronization property was considered, but it was decided instead to merely allow the parent shell process to adjust the process group of its child processes via *setpgid*(). Both timing constraints are now satisfied by having both the parent shell and the child attempt to adjust the process group of the child process; it does not matter which succeeds first.

Since it would be confusing to an application to have its process group change after it began executing (that is, after *exec*), and because the child process would already have adjusted its process group before this, the **[EACCES]** error was added to disallow this.

One non-obvious use of *setpgid*() is to allow a job control shell to return itself to its original process group (the one in effect when the job control shell was executed). A job control shell does this before returning control back to its parent when it is terminating or suspending itself as a way of restoring its job control "state" back to what its parent would expect. (Note that the original process group of the job control shell typically matches the process group of its parent, but this is not necessarily always the case.)

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getpgrp*( ), *setsid*( ), *tcsetpgrp*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setpgrp — set the process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t setpgrp(void);
```

DESCRIPTION

If the calling process is not already a session leader, *setpgrp*() sets the process group ID of the calling process to the process ID of the calling process. If *setpgrp*() creates a new session, then the new session has no controlling terminal.

The *setpgrp*() function has no effect when the calling process is a session leader.

RETURN VALUE

Upon completion, *setpgrp*() shall return the process group ID.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

It is unspecified whether this function behaves as *setpgid*(0,0) or *setsid*() unless the process is already a session leader. Therefore, applications are encouraged to use *setpgid*() or *setsid*() as appropriate.

RATIONALE

None.

FUTURE DIRECTIONS

The *setpgrp*() function may be removed in a future version.

SEE ALSO

exec, *fork*( ), *getpid*( ), *getsid*( ), *kill*( ), *setpgid*( ), *setsid*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setpriority — set the nice value

SYNOPSIS

#include <sys/resource.h>

int setpriority(int *which*, id_t *who*, int *nice*);

DESCRIPTION

Refer to *getpriority*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setprotoent — network protocol database functions

SYNOPSIS

```
#include <netdb.h>

void setprotoent(int stayopen);
```

DESCRIPTION

Refer to *endprotoent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setpwent — user database function

SYNOPSIS

```
#include <pwd.h>

void setpwent(void);
```

DESCRIPTION

Refer to *endpwent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setregid — set real and effective group IDs

SYNOPSIS

#include <unistd.h>

int setregid(gid_t *rgid*, gid_t *egid*);

DESCRIPTION

The *setregid*() function shall set the real and effective group IDs of the calling process.

If *rgid* is −1, the real group ID shall not be changed; if *egid* is −1, the effective group ID shall not be changed.

The real and effective group IDs may be set to different values in the same call.

Only a process with appropriate privileges can set the real group ID and the effective group ID to any valid value.

A non-privileged process can set either the real group ID to the saved set-group-ID from one of the *exec* family of functions, or the effective group ID to the saved set-group-ID or the real group ID.

If the real group ID is being set (*rgid* is not −1), or the effective group ID is being set to a value not equal to the real group ID, then the saved set-group-ID of the current process shall be set equal to the new effective group ID.

Any supplementary group IDs of the calling process remain unchanged.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error, and neither of the group IDs are changed.

ERRORS

The *setregid*() function shall fail if:

EINVAL

The value of the *rgid* or *egid* argument is invalid or out-of-range.

EPERM

The process does not have appropriate privileges and a change other than changing the real group ID to the saved set-group-ID, or changing the effective group ID to the real group ID or the saved set-group-ID, was requested.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If a non-privileged set-group-ID process sets its effective group ID to its real group ID, it can only set its effective group ID back to the previous value if *rgid* was −1 in the *setregid*() call, since the saved-group-ID is not changed in that case. If *rgid* was equal to the real group ID in the *setregid*() call, then the saved set-group-ID will also have been changed to the real user ID.

RATIONALE

Earlier versions of this standard did not specify whether the saved set-group-ID was affected by *setregid*() calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective group ID and saved set-group-ID to be the same as the real group ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective group ID. Privileged applications could already do this using just *setgid*(), but for non-privileged applications the only standard method available is to use this feature of *setregid*().

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setreuid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setreuid — set real and effective user IDs

SYNOPSIS

```
#include <unistd.h>

int setreuid(uid_t ruid, uid_t euid);
```

DESCRIPTION

The *setreuid*() function shall set the real and effective user IDs of the current process to the values specified by the *ruid* and *euid* arguments. If *ruid* or *euid* is −1, the corresponding effective or real user ID of the current process shall be left unchanged.

A process with appropriate privileges can set either ID to any value. An unprivileged process can only set the effective user ID if the *euid* argument is equal to either the real, effective, or saved user ID of the process.

If the real user ID is being set (*ruid* is not −1), or the effective user ID is being set to a value not equal to the real user ID, then the saved set-user-ID of the current process shall be set equal to the new effective user ID.

It is unspecified whether a process without appropriate privileges is permitted to change the real user ID to match the current effective user ID or saved set-user-ID of the process.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setreuid*() function shall fail if:

EINVAL
The value of the *ruid* or *euid* argument is invalid or out-of-range.

EPERM
The current process does not have appropriate privileges, and either an attempt was made to change the effective user ID to a value other than the real user ID or the saved set-user-ID or an attempt was made to change the real user ID to a value not permitted by the implementation.

The following sections are informative.

EXAMPLES

Setting the Effective User ID to the Real User ID

The following example sets the effective user ID of the calling process to the real user ID, so that files created later will be owned by the current user. It also sets the saved set-user-ID to the real user ID, so any future attempt to set the effective user ID back to its previous value will fail.

```
#include <unistd.h>
#include <sys/types.h>
...
setreuid(getuid(), getuid());
...
```

APPLICATION USAGE

None.

RATIONALE

Earlier versions of this standard did not specify whether the saved set-user-ID was affected by *setreuid*() calls. This version specifies common existing practice that constitutes an important security feature. The ability to set both the effective user ID and saved set-user-ID to be the same as the real user ID means that any security weakness in code that is executed after that point cannot result in malicious code being executed with the previous effective user ID. Privileged applications could already do this using just *setuid*(), but for non-privileged applications the only standard method available is to use this feature of *setreuid*().

FUTURE DIRECTIONS

None.

SEE ALSO

getegid( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setuid*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setrlimit — control maximum resource consumption

SYNOPSIS

#include <sys/resource.h>

int setrlimit(int *resource*, const struct rlimit **rlp*);

DESCRIPTION

Refer to *getrlimit*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setservent — network services database functions

SYNOPSIS

#include <netdb.h>

void setservent(int *stayopen*);

DESCRIPTION

Refer to *endservent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setsid — create session and set process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t setsid(void);
```

DESCRIPTION

The *setsid*() function shall create a new session, if the calling process is not a process group leader. Upon return the calling process shall be the session leader of this new session, shall be the process group leader of a new process group, and shall have no controlling terminal. The process group ID of the calling process shall be set equal to the process ID of the calling process. The calling process shall be the only process in the new process group and the only process in the new session.

RETURN VALUE

Upon successful completion, *setsid*() shall return the value of the new process group ID of the calling process. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *setsid*() function shall fail if:

EPERM
The calling process is already a process group leader, or the process group ID of a process other than the calling process matches the process ID of the calling process.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *setsid*() function is similar to the *setpgrp*() function of System V. System V, without job control, groups processes into process groups and creates new process groups via *setpgrp*(); only one process group may be part of a login session.

Job control allows multiple process groups within a login session. In order to limit job control actions so that they can only affect processes in the same login session, this volume of POSIX.1-2017 adds the concept of a session that is created via *setsid*(). The *setsid*() function also creates the initial process group contained in the session. Additional process groups can be created via the *setpgid*() function. A System V process group would correspond to a POSIX System Interfaces session containing a single POSIX process group. Note that this function requires that the calling process not be a process group leader. The usual way to ensure this is true is to create a new process with *fork*() and have it call *setsid*(). The *fork*() function guarantees that the process ID of the new process does not match any existing process group ID.

FUTURE DIRECTIONS

None.

SEE ALSO

getsid( ), *setpgid*( ), *setpgrp*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setsockopt — set the socket options

SYNOPSIS

```
#include <sys/socket.h>

int setsockopt(int socket, int level, int option_name,
   const void *option_value, socklen_t option_len);
```

DESCRIPTION

The *setsockopt*() function shall set the option specified by the *option_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option_value* argument for the socket associated with the file descriptor specified by the *socket* argument.

The *level* argument specifies the protocol level at which the option resides. To set options at the socket level, specify the *level* argument as SOL_SOCKET. To set options at other levels, supply the appropriate *level* identifier for the protocol controlling the option. For example, to indicate that an option is interpreted by the TCP (Transport Control Protocol), set *level* to IPPROTO_TCP as defined in the *<netinet/in.h>* header.

The *option_name* argument specifies a single option to set. It can be one of the socket-level options defined in **<sys_socket.h>** and described in *Section 2.10.16*, *Use of Options*. If *option_name* is equal to SO_RCVTIMEO or SO_SNDTIMEO and the implementation supports setting the option, it is unspecified whether the **struct timeval** pointed to by *option_value* is stored as provided by this function or is rounded up to align with the resolution of the clock being used. If *setsockopt*() is called with *option_name* equal to SO_ACCEPTCONN, SO_ERROR, or SO_TYPE, the behavior is unspecified.

RETURN VALUE

Upon successful completion, *setsockopt*() shall return 0. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setsockopt*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

EDOM
The send and receive timeout values are too big to fit into the timeout fields in the socket structure.

EINVAL
The specified option is invalid at the specified socket level or the socket has been shut down.

EISCONN
The socket is already connected, and a specified option cannot be set while the socket is connected.

ENOPROTOOPT
The option is not supported by the protocol.

ENOTSOCK
The *socket* argument does not refer to a socket.

The *setsockopt*() function may fail if:

ENOMEM
There was insufficient memory available for the operation to complete.

ENOBUFS
Insufficient resources are available in the system to complete the call.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *setsockopt*() function provides an application program with the means to control socket behavior. An application program can use *setsockopt*() to allocate buffer space, control timeouts, or permit socket data broadcasts. The <*sys/socket.h*> header defines the socket-level options available to *setsockopt*().

Options may exist at multiple protocol levels. The SO_ options are always present at the uppermost socket level.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10, *Sockets*, *bind*( ), *endprotoent*( ), *getsockopt*( ), *socket*( )

The Base Definitions volume of POSIX.1-2017, **<netinet_in.h>**, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setstate — switch pseudo-random number generator state arrays

SYNOPSIS

#include <stdlib.h>

char *setstate(char **state*);

DESCRIPTION

Refer to *initstate*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setuid — set user ID

SYNOPSIS

#include <unistd.h>

int setuid(uid_t *uid*);

DESCRIPTION

If the process has appropriate privileges, *setuid*() shall set the real user ID, effective user ID, and the saved set-user-ID of the calling process to *uid*.

If the process does not have appropriate privileges, but *uid* is equal to the real user ID or the saved set-user-ID, *setuid*() shall set the effective user ID to *uid*; the real user ID and saved set-user-ID shall remain unchanged.

The *setuid*() function shall not affect the supplementary group list in any way.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *setuid*() function shall fail, return −1, and set *errno* to the corresponding value if one or more of the following are true:

EINVAL
The value of the *uid* argument is invalid and not supported by the implementation.

EPERM
The process does not have appropriate privileges and *uid* does not match the real user ID or the saved set-user-ID.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The various behaviors of the *setuid*() and *setgid*() functions when called by non-privileged processes reflect the behavior of different historical implementations. For portability, it is recommended that new non-privileged applications use the *seteuid*() and *setegid*() functions instead.

The saved set-user-ID capability allows a program to regain the effective user ID established at the last *exec* call. Similarly, the saved set-group-ID capability allows a program to regain the effective group ID established at the last *exec* call. These capabilities are derived from System V. Without them, a program might have to run as superuser in order to perform the same functions, because superuser can write on the user's files. This is a problem because such a program can write on any user's files, and so must be carefully written to emulate the permissions of the calling process properly. In System V, these capabilities have traditionally been implemented only via the *setuid*() and *setgid*() functions for non-privileged processes. The fact that the behavior of those functions was different for privileged processes made them difficult to use. The POSIX.1-1990 standard defined the *setuid*() function to behave differently for privileged and unprivileged users. When the caller had appropriate privileges, the function set the real user ID, effective user ID, and saved set-user ID of the calling process on implementations that supported it. When the caller did not have appropriate privileges, the function set only the effective user ID, subject to permission checks. The former use is generally needed for utilities like *login* and *su*, which are not conforming applications and thus outside the scope of POSIX.1-2008. These utilities wish to change the user ID irrevocably to a new value, generally that of an unprivileged user. The latter use is needed for conforming applications that are installed with the set-user-ID bit and need to perform operations using the real user ID.

POSIX.1-2008 augments the latter functionality with a mandatory feature named _POSIX_SAVED_IDS. This feature permits a set-user-ID application to switch its effective user ID back and forth between the values of its *exec*-time real user ID and effective user ID. Unfortunately, the POSIX.1-1990 standard did not permit a conforming application using this feature to work properly when it happened to be executed with (implementation-defined) appropriate privileges. Furthermore, the application did not even have a means to tell whether it had this privilege. Since the saved set-user-ID feature is quite desirable for applications, as evidenced by the fact that NIST required it in FIPS 151-2, it has been mandated by POSIX.1-2008. However, there are implementors who have been reluctant to support it given the limitation described above.

The 4.3BSD system handles the problem by supporting separate functions: *setuid*() (which always sets both the real and effective user IDs, like *setuid*() in POSIX.1-2008 for privileged users), and *seteuid*() (which always sets just the effective user ID, like *setuid*() in POSIX.1-2008 for non-privileged users). This separation of functionality into distinct functions seems desirable. 4.3BSD does not support the saved set-user-ID feature. It supports similar functionality of switching the effective user ID back and forth via *setreuid*(), which permits reversing the real and effective user IDs. This model seems less desirable than the saved set-user-ID because the real user ID changes as a side-effect. The current 4.4BSD includes saved effective IDs and uses them for *seteuid*() and *setegid*() as described above. The *setreuid*() and *setregid*() functions will be deprecated or removed.

The solution here is:

* Require that all implementations support the functionality of the saved set-user-ID, which is set by the *exec* functions and by privileged calls to *setuid*().
* Add the *seteuid*() and *setegid*() functions as portable alternatives to *setuid*() and *setgid*() for non-privileged and privileged processes.

Historical systems have provided two mechanisms for a set-user-ID process to change its effective user ID to be the same as its real user ID in such a way that it could return to the original effective user ID: the use of the *setuid*() function in the presence of a saved set-user-ID, or the use of the BSD *setreuid*() function, which was able to swap the real and effective user IDs. The changes included in POSIX.1-2008 provide a new mechanism using *seteuid*() in conjunction with a saved set-user-ID. Thus, all implementations with the new *seteuid*() mechanism will have a saved set-user-ID for each process, and most of the behavior controlled by _POSIX_SAVED_IDS has been changed to agree with the case where the option was defined. The *kill*() function is an exception. Implementors of the new *seteuid*() mechanism will generally be required to maintain compatibility with the older mechanisms previously supported by their systems. However, compatibility with this use of *setreuid*() and with the _POSIX_SAVED_IDS behavior of *kill*() is unfortunately complicated. If an implementation with a saved set-user-ID allows a process to use *setreuid*() to swap its real and effective user IDs, but were to leave the saved set-user-ID unmodified, the process would then have an effective user ID equal to the original real user ID, and both real and saved set-user-ID would be equal to the original effective user ID. In that state, the real user would be unable to kill the process, even though the effective user ID of the process matches that of the real user, if the *kill*() behavior of _POSIX_SAVED_IDS was used. This is obviously not acceptable. The alternative choice, which is used in at least one implementation, is to change the saved set-user-ID to the effective user ID during most calls to *setreuid*(). The standard developers considered that alternative to be less correct than the retention of the old behavior of *kill*() in such systems. Current conforming applications shall accommodate either behavior from *kill*(), and there appears to be no strong reason for *kill*() to check the saved set-user-ID rather than the effective user ID.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *getegid*( ), *geteuid*( ), *getgid*( ), *getuid*( ), *setegid*( ), *seteuid*( ), *setgid*( ), *setregid*( ), *setreuid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setutxent — reset the user accounting database to the first entry

SYNOPSIS

```
#include <utmpx.h>

void setutxent(void);
```

DESCRIPTION

Refer to *endutxent*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

setvbuf — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

int setvbuf(FILE *restrict stream, char *restrict buf, int type,
   size_t size);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *setvbuf*() function may be used after the stream pointed to by *stream* is associated with an open file but before any other operation (other than an unsuccessful call to *setvbuf*()) is performed on the stream. The argument *type* determines how *stream* shall be buffered, as follows:

* {_IOFBF} shall cause input/output to be fully buffered.
* {_IOLBF} shall cause input/output to be line buffered.
* {_IONBF} shall cause input/output to be unbuffered.

If *buf* is not a null pointer, the array it points to may be used instead of a buffer allocated by *setvbuf*() and the argument *size* specifies the size of the array; otherwise, *size* may determine the size of a buffer allocated by the *setvbuf*() function. The contents of the array at any time are unspecified.

For information about streams, see *Section 2.5*, *Standard I/O Streams*.

RETURN VALUE

Upon successful completion, *setvbuf*() shall return 0. Otherwise, it shall return a non-zero value if an invalid value is given for *type* or if the request cannot be honored, and may set *errno* to indicate the error.

ERRORS

The *setvbuf*() function may fail if:

EBADF
The file descriptor underlying *stream* is not valid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

A common source of error is allocating buffer space as an ''automatic'' variable in a code block, and then failing to close the stream in the same block.

With *setvbuf*(), allocating a buffer of *size* bytes does not necessarily imply that all of *size* bytes are used for the buffer area.

Applications should note that many implementations only provide line buffering on input from terminal devices.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *setbuf*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shm_open — open a shared memory object (**REALTIME**)

SYNOPSIS

#include <sys/mman.h>

int shm_open(const char **name*, int *oflag*, mode_t *mode*);

DESCRIPTION

The *shm_open*() function shall establish a connection between a shared memory object and a file descriptor. It shall create an open file description that refers to the shared memory object and a file descriptor that refers to that open file description. The file descriptor shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*, and can be used by other functions to refer to that shared memory object. The *name* argument points to a string naming a shared memory object. It is unspecified whether the name appears in the file system and is visible to other functions that take pathnames as arguments. The *name* argument conforms to the construction rules for a pathname, except that the interpretation of <slash> characters other than the leading <slash> character in *name* is implementation-defined, and that the length limits for the *name* argument are implementation-defined and need not be the same as the pathname limits {PATH_MAX} and {NAME_MAX}. If *name* begins with the <slash> character, then processes calling *shm_open*() with the same value of *name* refer to the same shared memory object, as long as that name has not been removed. If *name* does not begin with the <slash> character, the effect is implementation-defined.

If successful, *shm_open*() shall return a file descriptor for the shared memory object. The open file description is new, and therefore the file descriptor does not share it with any other processes. It is unspecified whether the file offset is set. The FD_CLOEXEC file descriptor flag associated with the new file descriptor is set.

The file status flags and file access modes of the open file description are according to the value of *oflag*. The *oflag* argument is the bitwise-inclusive OR of the following flags defined in the *<fcntl.h>* header. Applications specify exactly one of the first two values (access modes) below in the value of *oflag*:

O_RDONLY	Open for read access only.
O_RDWR	Open for read or write access.

Any combination of the remaining flags may be specified in the value of *oflag*:

O_CREAT	If the shared memory object exists, this flag has no effect, except as noted under O_EXCL below. Otherwise, the shared memory object is created. The user ID of the shared memory object shall be set to the effective user ID of the process. The group ID of the shared memory object shall be set to the effective group ID of the process; however, if the *name* argument is visible in the file system, the group ID may be set to the group ID of the containing directory. The permission bits of the shared memory object shall be set to the value of the *mode* argument except those set in the file mode creation mask of the process. When bits in *mode* other than the file permission bits are set, the effect is unspecified. The *mode* argument does not affect whether the shared memory object is opened for reading, for writing, or for both. The shared memory object has a size of zero.
O_EXCL	If O_EXCL and O_CREAT are set, *shm_open*() fails if the shared memory object exists. The check for the existence of the shared memory object and the creation of the object if it does not exist is atomic with respect to other processes executing *shm_open*() naming the same shared memory object with O_EXCL and O_CREAT set. If O_EXCL is set and O_CREAT is not set, the result is undefined.
O_TRUNC	If the shared memory object exists, and it is successfully opened O_RDWR, the object shall be truncated to zero length and the mode and owner shall be unchanged by this function call. The result of using O_TRUNC with O_RDONLY is undefined.

When a shared memory object is created, the state of the shared memory object, including all data associated with the shared memory object, persists until the shared memory object is unlinked and all other references are gone. It is unspecified whether the name and shared memory object state remain valid after a system reboot.

RETURN VALUE

Upon successful completion, the *shm_open*() function shall return a non-negative integer representing the file descriptor. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *shm_open*() function shall fail if:

EACCES
The shared memory object exists and the permissions specified by *oflag* are denied, or the shared memory object does not exist and permission to create the shared memory object is denied, or O_TRUNC is specified and write permission is denied.

EEXIST
O_CREAT and O_EXCL are set and the named shared memory object already exists.

EINTR
The *shm_open*() operation was interrupted by a signal.

EINVAL
The *shm_open*() operation is not supported for the given name.

EMFILE
All file descriptors available to the process are currently open.

ENFILE
Too many shared memory objects are currently open in the system.

ENOENT
O_CREAT is not set and the named shared memory object does not exist.

ENOSPC
There is insufficient space for the creation of the new shared memory object.

The *shm_open*() function may fail if:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems.

The following sections are informative.

EXAMPLES

Creating and Mapping a Shared Memory Object

The following code segment demonstrates the use of *shm_open*() to create a shared memory object which is then sized using *ftruncate*() before being mapped into the process address space using *mmap*():

```
#include <unistd.h>
#include <sys/mman.h>
...
#define MAX_LEN 10000
struct region {        /* Defines "structure" of shared memory */
    int len;
    char buf[MAX_LEN];
};
```

```
struct region *rptr;
int fd;

/* Create shared memory object and set its size */

fd = shm_open("/myregion", O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);
if (fd == -1)
    /* Handle error */;

if (ftruncate(fd, sizeof(struct region)) == -1)
    /* Handle error */;

/* Map shared memory object */

rptr = mmap(NULL, sizeof(struct region),
       PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0);
if (rptr == MAP_FAILED)
    /* Handle error */;

/* Now we can refer to mapped region using fields of rptr;
   for example, rptr->len */
...
```

APPLICATION USAGE

None.

RATIONALE

When the Memory Mapped Files option is supported, the normal *open*() call is used to obtain a descriptor to a file to be mapped according to existing practice with *mmap*(). When the Shared Memory Objects option is supported, the *shm_open*() function shall obtain a descriptor to the shared memory object to be mapped.

There is ample precedent for having a file descriptor represent several types of objects. In the POSIX.1-1990 standard, a file descriptor can represent a file, a pipe, a FIFO, a tty, or a directory. Many implementations simply have an operations vector, which is indexed by the file descriptor type and does very different operations. Note that in some cases the file descriptor passed to generic operations on file descriptors is returned by *open*() or *creat*() and in some cases returned by alternate functions, such as *pipe*(). The latter technique is used by *shm_open*().

Note that such shared memory objects can actually be implemented as mapped files. In both cases, the size can be set after the open using *ftruncate*(). The *shm_open*() function itself does not create a shared object of a specified size because this would duplicate an extant function that set the size of an object referenced by a file descriptor.

On implementations where memory objects are implemented using the existing file system, the *shm_open*() function may be implemented using a macro that invokes *open*(), and the *shm_unlink*() function may be implemented using a macro that invokes *unlink*().

For implementations without a permanent file system, the definition of the name of the memory objects is allowed not to survive a system reboot. Note that this allows systems with a permanent file system to implement memory objects as data structures internal to the implementation as well.

On implementations that choose to implement memory objects using memory directly, a *shm_open*() followed by an *ftruncate*() and *close*() can be used to preallocate a shared memory area and to set the size of that preallocation. This may be necessary for systems without virtual memory hardware support in order to ensure that the memory is contiguous.

The set of valid open flags to *shm_open*() was restricted to O_RDONLY, O_RDWR, O_CREAT, and O_TRUNC because these could be easily implemented on most memory mapping systems. This volume of POSIX.1-2017 is silent on the results if the implementation cannot supply the requested file access because of implementation-defined reasons, including hardware ones.

The error conditions **[EACCES]** and **[ENOTSUP]** are provided to inform the application that the implementation cannot complete a request.

[EACCES] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode because it conflicts with another requested mode. An example might be that an application desires to open a memory object two times, mapping different areas with different access modes. If the implementation cannot map a single area into a process space in two places, which would be required if different access modes were required for the two areas, then the implementation may inform the application at the time of the second open.

[ENOTSUP] indicates for implementation-defined reasons, probably hardware-related, that the implementation cannot comply with a requested mode at all. An example would be that the hardware of the implementation cannot support write-only shared memory areas.

On all implementations, it may be desirable to restrict the location of the memory objects to specific file systems for performance (such as a RAM disk) or implementation-defined reasons (shared memory supported directly only on certain file systems). The *shm_open*() function may be used to enforce these restrictions. There are a number of methods available to the application to determine an appropriate name of the file or the location of an appropriate directory. One way is from the environment via *getenv*(). Another would be from a configuration file.

This volume of POSIX.1-2017 specifies that memory objects have initial contents of zero when created. This is consistent with current behavior for both files and newly allocated memory. For those implementations that use physical memory, it would be possible that such implementations could simply use available memory and give it to the process uninitialized. This, however, is not consistent with standard behavior for the uninitialized data area, the stack, and of course, files. Finally, it is highly desirable to set the allocated memory to zero for security reasons. Thus, initializing memory objects to zero is required.

FUTURE DIRECTIONS

A future version might require the *shm_open*() and *shm_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *close*( ), *dup*( ), *exec*, *fcntl*( ), *mmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_unlink*( ), *umask*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG
This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME
shm_unlink — remove a shared memory object (**REALTIME**)

SYNOPSIS
#include <sys/mman.h>

int shm_unlink(const char **name*);

DESCRIPTION
The *shm_unlink*() function shall remove the name of the shared memory object named by the string pointed to by *name*.

If one or more references to the shared memory object exist when the object is unlinked, the name shall be removed before *shm_unlink*() returns, but the removal of the memory object contents shall be postponed until all open and map references to the shared memory object have been removed.

Even if the object continues to exist after the last *shm_unlink*(), reuse of the name shall subsequently cause *shm_open*() to behave as if no shared memory object of this name exists (that is, *shm_open*() will fail if O_CREAT is not set, or will create a new shared memory object if O_CREAT is set).

RETURN VALUE
Upon successful completion, a value of zero shall be returned. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error. If −1 is returned, the named shared memory object shall not be changed by this function call.

ERRORS
The *shm_unlink*() function shall fail if:

EACCES
Permission is denied to unlink the named shared memory object.

ENOENT
The named shared memory object does not exist.

The *shm_unlink*() function may fail if:

ENAMETOOLONG
The length of the *name* argument exceeds {_POSIX_PATH_MAX} on systems that do not support the XSI option or exceeds {_XOPEN_PATH_MAX} on XSI systems, or has a pathname component that is longer than {_POSIX_NAME_MAX} on systems that do not support the XSI option or longer than {_XOPEN_NAME_MAX} on XSI systems. A call to *shm_unlink*() with a *name* argument that contains the same shared memory object name as was previously used in a successful *shm_open*() call shall not give an **[ENAMETOOLONG]** error.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
Names of memory objects that were allocated with *open*() are deleted with *unlink*() in the usual fashion. Names of memory objects that were allocated with *shm_open*() are deleted with *shm_unlink*(). Note that the actual memory object is not destroyed until the last close and unmap on it have occurred if it was already in use.

RATIONALE
None.

FUTURE DIRECTIONS

A future version might require the *shm_open*() and *shm_unlink*() functions to have semantics similar to normal file system operations.

SEE ALSO

close( ), *mmap*( ), *munmap*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( )

The Base Definitions volume of POSIX.1-2017, **<sys_mman.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shmat — XSI shared memory attach operation

SYNOPSIS

#include <sys/shm.h>

void *shmat(int *shmid*, const void **shmaddr*, int *shmflg*);

DESCRIPTION

The *shmat*() function operates on XSI shared memory (see the Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *shmat*() function attaches the shared memory segment associated with the shared memory identifier specified by *shmid* to the address space of the calling process. The segment is attached at the address specified by one of the following criteria:

* If *shmaddr* is a null pointer, the segment is attached at the first available address as selected by the system.

* If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is non-zero, the segment is attached at the address given by (*shmaddr* −((*uintptr_t*)*shmaddr* %SHMLBA)). The character **'%'** is the C-language remainder operator.

* If *shmaddr* is not a null pointer and (*shmflg* &SHM_RND) is 0, the segment is attached at the address given by *shmaddr*.

* The segment is attached for reading if (*shmflg* &SHM_RDONLY) is non-zero and the calling process has read permission; otherwise, if it is 0 and the calling process has read and write permission, the segment is attached for reading and writing.

RETURN VALUE

Upon successful completion, *shmat*() shall increment the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return the segment's start address. Also, the *shm_atime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

Otherwise, the shared memory segment shall not be attached, *shmat*() shall return (**void** *)−1, and *errno* shall be set to indicate the error.

ERRORS

The *shmat*() function shall fail if:

EACCES
Operation permission is denied to the calling process; see *Section 2.7*, *XSI Interprocess Communication*.

EINVAL
The value of *shmid* is not a valid shared memory identifier, the *shmaddr* is not a null pointer, and the value of (*shmaddr* −((*uintptr_t*)*shmaddr* %SHMLBA)) is an illegal address for attaching shared memory; or the *shmaddr* is not a null pointer, (*shmflg* &SHM_RND) is 0, and the value of *shmaddr* is an illegal address for attaching shared memory.

EMFILE
The number of shared memory segments attached to the calling process would exceed the system-imposed limit.

ENOMEM
The available data space is not large enough to accommodate the shared memory segment.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *exec*, *exit*( ), *fork*( ), *shmctl*( ), *shmdt*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*, **<sys_shm.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shmctl — XSI shared memory control operations

SYNOPSIS

```
#include <sys/shm.h>

int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

DESCRIPTION

The *shmctl*() function operates on XSI shared memory (see the Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *shmctl*() function provides a variety of shared memory control operations as specified by *cmd*. The following values for *cmd* are available:

IPC_STAT
: Place the current value of each member of the **shmid_ds** data structure associated with *shmid* into the structure pointed to by *buf*. The contents of the structure are defined in *<sys/shm.h>*.

IPC_SET
: Set the value of the following members of the **shmid_ds** data structure associated with *shmid* to the corresponding value found in the structure pointed to by *buf*:

shm_perm.uid	
shm_perm.gid	
shm_perm.mode	Low-order nine bits.

Also, the *shm_ctime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

IPC_SET can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

IPC_RMID
: Remove the shared memory identifier specified by *shmid* from the system and destroy the shared memory segment and **shmid_ds** data structure associated with it. IPC_RMID can only be executed by a process that has an effective user ID equal to either that of a process with appropriate privileges or to the value of *shm_perm.cuid* or *shm_perm.uid* in the **shmid_ds** data structure associated with *shmid*.

RETURN VALUE

Upon successful completion, *shmctl*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *shmctl*() function shall fail if:

EACCES

The argument *cmd* is equal to IPC_STAT and the calling process does not have read permission; see *Section 2.7*, *XSI Interprocess Communication*.

EINVAL

The value of *shmid* is not a valid shared memory identifier, or the value of *cmd* is not a valid command.

EPERM
The argument *cmd* is equal to IPC_RMID or IPC_SET and the effective user ID of the calling process is not equal to that of a process with appropriate privileges and it is not equal to the value of *shm_perm.cuid* or *shm_perm.uid* in the data structure associated with *shmid*.

The *shmctl*() function may fail if:

EOVERFLOW
The *cmd* argument is IPC_STAT and the *gid* or *uid* value is too large to be stored in the structure pointed to by the *buf* argument.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *shmat*( ), *shmdt*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*, **<sys_shm.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shmdt — XSI shared memory detach operation

SYNOPSIS

```
#include <sys/shm.h>

int shmdt(const void *shmaddr);
```

DESCRIPTION

The *shmdt*() function operates on XSI shared memory (see the Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *shmdt*() function detaches the shared memory segment located at the address specified by *shmaddr* from the address space of the calling process.

RETURN VALUE

Upon successful completion, *shmdt*() shall decrement the value of *shm_nattch* in the data structure associated with the shared memory ID of the attached shared memory segment and return 0. Also, the *shm_dtime* timestamp shall be set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

Otherwise, the shared memory segment shall not be detached, *shmdt*() shall return −1, and *errno* shall be set to indicate the error.

ERRORS

The *shmdt*() function shall fail if:

EINVAL
The value of *shmaddr* is not the data segment start address of a shared memory segment.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *exec*, *exit*( ), *fork*( ), *shmat*( ), *shmctl*( ), *shmget*( ), *shm_open*( ), *shm_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*, **<sys_shm.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shmget — get an XSI shared memory segment

SYNOPSIS

#include <sys/shm.h>

int shmget(key_t *key*, size_t *size*, int *shmflg*);

DESCRIPTION

The *shmget*() function operates on XSI shared memory (see the Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*). It is unspecified whether this function interoperates with the realtime interprocess communication facilities defined in *Section 2.8*, *Realtime*.

The *shmget*() function shall return the shared memory identifier associated with *key*.

A shared memory identifier, associated data structure, and shared memory segment of at least *size* bytes (see *<sys/shm.h>*) are created for *key* if one of the following is true:

* The argument *key* is equal to IPC_PRIVATE.
* The argument *key* does not already have a shared memory identifier associated with it and (*shmflg* &IPC_CREAT) is non-zero.

Upon creation, the data structure associated with the new shared memory identifier shall be initialized as follows:

* The values of *shm_perm.cuid*, *shm_perm.uid*, *shm_perm.cgid*, and *shm_perm.gid* are set to the effective user ID and effective group ID, respectively, of the calling process.
* The low-order nine bits of *shm_perm.mode* are set to the low-order nine bits of *shmflg*.
* The value of *shm_segsz* is set to the value of *size*.
* The values of *shm_lpid*, *shm_nattch*, *shm_atime*, and *shm_dtime* are set to 0.
* The value of *shm_ctime* is set to the current time, as described in *Section 2.7.1*, *IPC General Description*.

When the shared memory segment is created, it shall be initialized with all zero values.

RETURN VALUE

Upon successful completion, *shmget*() shall return a non-negative integer, namely a shared memory identifier; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *shmget*() function shall fail if:

EACCES
A shared memory identifier exists for *key* but operation permission as specified by the low-order nine bits of *shmflg* would not be granted; see *Section 2.7*, *XSI Interprocess Communication*.

EEXIST
A shared memory identifier exists for the argument *key* but (*shmflg* &IPC_CREAT) &&(*shmflg* &IPC_EXCL) is non-zero.

EINVAL
A shared memory segment is to be created and the value of size is less than the system-imposed minimum or greater than the system-imposed maximum.

EINVAL
No shared memory segment is to be created and a shared memory segment exists for *key* but the size of the segment associated with it is less than *size*.

ENOENT
A shared memory identifier does not exist for the argument *key* and (*shmflg* &IPC_CREAT) is 0.

ENOMEM
A shared memory identifier and associated shared memory segment are to be created, but the amount of available physical memory is not sufficient to fill the request.

ENOSPC
A shared memory identifier is to be created, but the system-imposed limit on the maximum number of allowed shared memory identifiers system-wide would be exceeded.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The POSIX Realtime Extension defines alternative interfaces for interprocess communication. Application developers who need to use IPC should design their applications so that modules using the IPC routines described in *Section 2.7*, *XSI Interprocess Communication* can be easily modified to use the alternative interfaces.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.7, *XSI Interprocess Communication*, *Section 2.8*, *Realtime*, *ftok*( ), *shmat*( ), *shmctl*( ), *shmdt*( ), *shm_open*( ), *shm_unlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 3.346*, *Shared Memory Object*, **<sys_shm.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

shutdown — shut down socket send and receive operations

SYNOPSIS

```
#include <sys/socket.h>

int shutdown(int socket, int how);
```

DESCRIPTION

The *shutdown*() function shall cause all or part of a full-duplex connection on the socket associated with the file descriptor *socket* to be shut down.

The *shutdown*() function takes the following arguments:

socket	Specifies the file descriptor of the socket.
how	Specifies the type of shutdown. The values are as follows:

SHUT_RD	Disables further receive operations.
SHUT_WR	Disables further send operations.
SHUT_RDWR	Disables further send and receive operations.

The *shutdown*() function disables subsequent send and/or receive operations on a socket, depending on the value of the *how* argument.

RETURN VALUE

Upon successful completion, *shutdown*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *shutdown*() function shall fail if:

EBADF
The *socket* argument is not a valid file descriptor.

EINVAL
The *how* argument is invalid.

ENOTCONN
The socket is not connected.

ENOTSOCK
The *socket* argument does not refer to a socket.

The *shutdown*() function may fail if:

ENOBUFS
Insufficient resources were available in the system to perform the operation.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

getsockopt( ), *pselect*( ), *read*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendto*( ), *setsockopt*( ), *socket*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigaction — examine and change a signal action

SYNOPSIS

```
#include <signal.h>

int sigaction(int sig, const struct sigaction *restrict act,
    struct sigaction *restrict oact);
```

DESCRIPTION

The *sigaction*() function allows the calling process to examine and/or specify the action to be associated with a specific signal. The argument *sig* specifies the signal; acceptable values are defined in *<signal.h>*.

The structure **sigaction**, used to describe an action to be taken, is defined in the *<signal.h>* header to include at least the following members:

center box tab(!); cB | cB | cB lw(1.5i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ void(*) (int)!sa_handler!T{ Pointer to a signal-catching function or one of the macros SIG_IGN or SIG_DFL. T} sigset_t!sa_mask!T{ Additional set of signals to be blocked during execution of signal-catching function. T} int!sa_flags!T{ Special flags to affect behavior of signal. T} T{ void(*) (int,
   siginfo_t *, void *) T}!sa_sigaction!Pointer to a signal-catching function.

The storage occupied by *sa_handler* and *sa_sigaction* may overlap, and a conforming application shall not use both simultaneously.

If the argument *act* is not a null pointer, it points to a structure specifying the action to be associated with the specified signal. If the argument *oact* is not a null pointer, the action previously associated with the signal is stored in the location pointed to by the argument *oact*. If the argument *act* is a null pointer, signal handling is unchanged; thus, the call can be used to enquire about the current handling of a given signal. The SIGKILL and SIGSTOP signals shall not be added to the signal mask using this mechanism; this restriction shall be enforced by the system without causing an error to be indicated.

If the SA_SIGINFO flag (see below) is cleared in the *sa_flags* field of the **sigaction** structure, the *sa_handler* field identifies the action to be associated with the specified signal. If the SA_SIGINFO flag is set in the *sa_flags* field, the *sa_sigaction* field specifies a signal-catching function.

The *sa_flags* field can be used to modify the behavior of the specified signal.

The following flags, defined in the *<signal.h>* header, can be set in *sa_flags*:

SA_NOCLDSTOP
: Do not generate SIGCHLD when children stop or stopped children continue.

  If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in *sa_flags*, and the implementation supports the SIGCHLD signal, then a SIGCHLD signal shall be generated for the calling process whenever any of its child processes stop and a SIGCHLD signal may be generated for the calling process whenever any of its stopped child processes are continued. If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is set in *sa_flags*, then the implementation shall not generate a SIGCHLD signal in this way.

SA_ONSTACK
: If set and an alternate signal stack has been declared with *sigaltstack*(), the signal shall be delivered to the calling process on that stack. Otherwise, the signal shall be delivered on the current stack.

SA_RESETHAND
: If set, the disposition of the signal shall be reset to SIG_DFL and the SA_SIGINFO flag shall be cleared on entry to the signal handler.

Note: SIGILL and SIGTRAP cannot be automatically reset when delivered; the system silently enforces this restriction.

Otherwise, the disposition of the signal shall not be modified on entry to the signal handler.

In addition, if this flag is set, *sigaction*() may behave as if the SA_NODEFER flag were also set.

SA_RESTART This flag affects the behavior of interruptible functions; that is, those specified to fail with *errno* set to **[EINTR]**. If set, and a function specified as interruptible is interrupted by this signal, the function shall restart and shall not fail with **[EINTR]** unless otherwise specified. If an interruptible function which uses a timeout is restarted, the duration of the timeout following the restart is set to an unspecified value that does not exceed the original timeout value. If the flag is not set, interruptible functions interrupted by this signal shall fail with *errno* set to **[EINTR]**.

SA_SIGINFO If cleared and the signal is caught, the signal-catching function shall be entered as:

```
void func(int signo);
```

where *signo* is the only argument to the signal-catching function. In this case, the application shall use the *sa_handler* member to describe the signal-catching function and the application shall not modify the *sa_sigaction* member.

If SA_SIGINFO is set and the signal is caught, the signal-catching function shall be entered as:

```
void func(int signo, siginfo_t *info, void *context);
```

where two additional arguments are passed to the signal-catching function. The second argument shall point to an object of type **siginfo_t** explaining the reason why the signal was generated; the third argument can be cast to a pointer to an object of type **ucontext_t** to refer to the receiving thread's context that was interrupted when the signal was delivered. In this case, the application shall use the *sa_sigaction* member to describe the signal-catching function and the application shall not modify the *sa_handler* member.

The *si_signo* member contains the system-generated signal number.

The *si_errno* member may contain implementation-defined additional error information; if non-zero, it contains an error number identifying the condition that caused the signal to be generated.

The *si_code* member contains a code identifying the cause of the signal, as described in *Section 2.4.3*, *Signal Actions*.

SA_NOCLDWAIT
If *sig* does not equal SIGCHLD, the behavior is unspecified. Otherwise, the behavior of the SA_NOCLDWAIT flag is as specified in *Consequences of Process Termination*.

SA_NODEFER If set and *sig* is caught, *sig* shall not be added to the thread's signal mask on entry to the signal handler unless it is included in *sa_mask*. Otherwise, *sig* shall always be added to the thread's signal mask on entry to the signal handler.

When a signal is caught by a signal-catching function installed by *sigaction*(), a new signal mask is calculated and installed for the duration of the signal-catching function (or until a call to either *sigprocmask*() or *sigsuspend*() is made). This mask is formed by taking the union of the current signal mask and the value of the *sa_mask* for the signal being delivered, and unless SA_NODEFER or SA_RESETHAND is set, then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it shall remain installed until another action is explicitly requested (by another call to *sigaction*()), until the SA_RESETHAND flag causes resetting of the handler, or until one of the *exec* functions is called.

If the previous action for *sig* had been established by *signal*(), the values of the fields returned in the structure pointed to by *oact* are unspecified, and in particular *oact*->*sa_handler* is not necessarily the same value passed to *signal*(). However, if a pointer to the same structure or a copy thereof is passed to a subsequent call to *sigaction*() via the *act* argument, handling of the signal shall be as if the original call to *signal*() were repeated.

If *sigaction*() fails, no new signal handler is installed.

It is unspecified whether an attempt to set the action for a signal that cannot be caught or ignored to SIG_DFL is ignored or causes an error to be returned with *errno* set to **[EINVAL]**.

If SA_SIGINFO is not set in *sa_flags*, then the disposition of subsequent occurrences of *sig* when it is already pending is implementation-defined; the signal-catching function shall be invoked with a single argument. If SA_SIGINFO is set in *sa_flags*, then subsequent occurrences of *sig* generated by *sigqueue*() or as a result of any signal-generating function that supports the specification of an application-defined value (when *sig* is already pending) shall be queued in FIFO order until delivered or accepted; the signal-catching function shall be invoked with three arguments. The application specified value is passed to the signal-catching function as the *si_value* member of the **siginfo_t** structure.

The result of the use of *sigaction*() and a *sigwait*() function concurrently within a process on the same signal is unspecified.

RETURN VALUE

Upon successful completion, *sigaction*() shall return 0; otherwise, −1 shall be returned, *errno* shall be set to indicate the error, and no new signal-catching function shall be installed.

ERRORS

The *sigaction*() function shall fail if:

EINVAL
The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

The *sigaction*() function may fail if:

EINVAL
An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

In addition, on systems that do not support the XSI option, the *sigaction*() function may fail if the SA_SIGINFO flag is set in the *sa_flags* field of the **sigaction** structure for a signal not in the range SIGRTMIN to SIGRTMAX.

The following sections are informative.

EXAMPLES

Establishing a Signal Handler

The following example demonstrates the use of *sigaction*() to establish a handler for the SIGINT signal.

```
#include <signal.h>

static void handler(int signum)
{
   /* Take appropriate actions for signal delivery */
}

int main()
{
   struct sigaction sa;

   sa.sa_handler = handler;
   sigemptyset(&sa.sa_mask);
   sa.sa_flags = SA_RESTART; /* Restart functions if
                    interrupted by handler */
   if (sigaction(SIGINT, &sa, NULL) == -1)
       /* Handle error */;

   /* Further code */
}
```

APPLICATION USAGE

The *sigaction*() function supersedes the *signal*() function, and should be used in preference. In particular, *sigaction*() and *signal*() should not be used in the same process to control the same signal. The behavior of async-signal-safe functions, as defined in their respective DESCRIPTION sections, is as specified by this volume of POSIX.1-2017, regardless of invocation from a signal-catching function. This is the only intended meaning of the statement that async-signal-safe functions may be used in signal-catching functions without restrictions. Applications must still consider all effects of such functions on such things as data structures, files, and process state. In particular, application developers need to consider the restrictions on interactions when interrupting *sleep*() and interactions among multiple handles for a file description. The fact that any specific function is listed as async-signal-safe does not necessarily mean that invocation of that function from a signal-catching function is recommended.

In order to prevent errors arising from interrupting non-async-signal-safe function calls, applications should protect calls to these functions either by blocking the appropriate signals or through the use of some programmatic semaphore (see *semget*( ), *sem_init*( ), *sem_open*( ), and so on). Note in particular that even the "safe" functions may modify *errno*; the signal-catching function, if not executing as an independent thread, should save and restore its value in order to avoid the possibility that delivery of a signal in between an error return from a function that sets *errno* and the subsequent examination of *errno* could result in the signal-catching function changing the value of *errno*. Naturally, the same principles apply to the async-signal-safety of application routines and asynchronous data access. Note that *longjmp*() and *siglongjmp*() are not in the list of async-signal-safe functions. This is because the code executing after *longjmp*() and *siglongjmp*() can call any unsafe functions with the same danger as calling those unsafe functions directly from the signal handler. Applications that use *longjmp*() and *siglongjmp*() from within signal handlers require rigorous protection in order to be portable. Many of the other functions that are excluded from the list are traditionally implemented using either *malloc*() or *free*() functions or the standard I/O library, both of which traditionally use data structures in a non-async-signal-safe manner. Since any combination of different functions using a common data structure can cause async-signal-safety problems, this volume of POSIX.1-2017 does not define the behavior when any unsafe function is called in a signal handler that interrupts an unsafe function.

Usually, the signal is executed on the stack that was in effect before the signal was delivered. An alternate stack may be specified to receive a subset of the signals being caught.

When the signal handler returns, the receiving thread resumes execution at the point it was interrupted unless the signal handler makes other arrangements. If *longjmp*() or *_longjmp*() is used to leave the signal handler, then the signal mask must be explicitly restored.

This volume of POSIX.1-2017 defines the third argument of a signal handling function when SA_SIGINFO is set as a **void *** instead of a **ucontext_t ***, but without requiring type checking. New applications should explicitly cast the third argument of the signal handling function to **ucontext_t ***.

The BSD optional four argument signal handling function is not supported by this volume of POSIX.1-2017. The BSD declaration would be:

```
    void handler(int sig, int code, struct sigcontext *scp,
        char *addr);
```

where *sig* is the signal number, *code* is additional information on certain signals, *scp* is a pointer to the **sigcontext** structure, and *addr* is additional address information. Much the same information is available in the objects pointed to by the second argument of the signal handler specified when SA_SIGINFO is set.

Since the *sigaction*() function is allowed but not required to set SA_NODEFER when the application sets the SA_RESETHAND flag, applications which depend on the SA_RESETHAND functionality for the newly installed signal handler must always explicitly set SA_NODEFER when they set SA_RESETHAND in order to be portable.

See also the rationale for Realtime Signal Generation and Delivery in the Rationale (Informative) volume of POSIX.1-2017, *Section B.2.4.2*, *Signal Generation and Delivery*.

RATIONALE

Although this volume of POSIX.1-2017 requires that signals that cannot be ignored shall not be added to the signal mask when a signal-catching function is entered, there is no explicit requirement that subsequent calls to *sigaction*() reflect this in the information returned in the *oact* argument. In other words, if SIGKILL is included in the *sa_mask* field of *act*, it is unspecified whether or not a subsequent call to *sigaction*() returns with SIGKILL included in the *sa_mask* field of *oact*.

The SA_NOCLDSTOP flag, when supplied in the *act*->*sa_flags* parameter, allows overloading SIGCHLD with the System V semantics that each SIGCLD signal indicates a single terminated child. Most conforming applications that catch SIGCHLD are expected to install signal-catching functions that repeatedly call the *waitpid*() function with the WNOHANG flag set, acting on each child for which status is returned, until *waitpid*() returns zero. If stopped children are not of interest, the use of the SA_NOCLDSTOP flag can prevent the overhead from invoking the signal-catching routine when they stop.

Some historical implementations also define other mechanisms for stopping processes, such as the *ptrace*() function. These implementations usually do not generate a SIGCHLD signal when processes stop due to this mechanism; however, that is beyond the scope of this volume of POSIX.1-2017.

This volume of POSIX.1-2017 requires that calls to *sigaction*() that supply a NULL *act* argument succeed, even in the case of signals that cannot be caught or ignored (that is, SIGKILL or SIGSTOP). The System V *signal*() and BSD *sigvec*() functions return **[EINVAL]** in these cases and, in this respect, their behavior varies from *sigaction*().

This volume of POSIX.1-2017 requires that *sigaction*() properly save and restore a signal action set up by the ISO C standard *signal*() function. However, there is no guarantee that the reverse is true, nor could there be given the greater amount of information conveyed by the **sigaction** structure. Because of this, applications should avoid using both functions for the same signal in the same process. Since this cannot always be avoided in case of general-purpose library routines, they should always be implemented with *sigaction*().

It was intended that the *signal*() function should be implementable as a library routine using *sigaction*().

The POSIX Realtime Extension extends the *sigaction*() function as specified by the POSIX.1-1990 standard to allow the application to request on a per-signal basis via an additional signal action flag that the extra parameters, including the application-defined signal value, if any, be passed to the signal-catching function.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *exec*, *_Exit*( ), *kill*( ), *_longjmp*( ), *longjmp*( ), *pthread_sigmask*( ), *raise*( ), *semget*( ), *sem_init*( ), *sem_open*( ), *sigaddset*( ), *sigaltstack*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *signal*( ), *sigsuspend*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigaddset — add a signal to a signal set

SYNOPSIS

```
#include <signal.h>

int sigaddset(sigset_t *set, int signo);
```

DESCRIPTION

The *sigaddset*() function adds the individual signal specified by the *signo* to the signal set pointed to by *set*.

Applications shall call either *sigemptyset*() or *sigfillset*() at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigismember*(), *sigpending*(), *sigprocmask*(), *sigsuspend*(), *sigtimedwait*(), *sigwait*(), or *sigwaitinfo*(), the results are undefined.

RETURN VALUE

Upon successful completion, *sigaddset*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *sigaddset*() function may fail if:

EINVAL
The value of the *signo* argument is an invalid or unsupported signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pthread_sigmask*( ), *sigaction*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigaltstack — set and get signal alternate stack context

SYNOPSIS

```
#include <signal.h>

int sigaltstack(const stack_t *restrict ss, stack_t *restrict oss);
```

DESCRIPTION

The *sigaltstack*() function allows a process to define and examine the state of an alternate stack for signal handlers for the current thread. Signals that have been explicitly declared to execute on the alternate stack shall be delivered on the alternate stack.

If *ss* is not a null pointer, it points to a **stack_t** structure that specifies the alternate signal stack that shall take effect upon return from *sigaltstack*(). The *ss_flags* member specifies the new stack state. If it is set to SS_DISABLE, the stack is disabled and *ss_sp* and *ss_size* are ignored. Otherwise, the stack shall be enabled, and the *ss_sp* and *ss_size* members specify the new address and size of the stack.

The range of addresses starting at *ss_sp* up to but not including *ss_sp*+*ss_size* is available to the implementation for use as the stack. This function makes no assumptions regarding which end is the stack base and in which direction the stack grows as items are pushed.

If *oss* is not a null pointer, upon successful completion it shall point to a **stack_t** structure that specifies the alternate signal stack that was in effect prior to the call to *sigaltstack*(). The *ss_sp* and *ss_size* members specify the address and size of that stack. The *ss_flags* member specifies the stack's state, and may contain one of the following values:

SS_ONSTACK
: The process is currently executing on the alternate signal stack. Attempts to modify the alternate signal stack while the process is executing on it fail. This flag shall not be modified by processes.

SS_DISABLE
: The alternate signal stack is currently disabled.

The value SIGSTKSZ is a system default specifying the number of bytes that would be used to cover the usual case when manually allocating an alternate stack area. The value MINSIGSTKSZ is defined to be the minimum stack size for a signal handler. In computing an alternate stack size, a program should add that amount to its stack requirements to allow for the system implementation overhead. The constants SS_ONSTACK, SS_DISABLE, SIGSTKSZ, and MINSIGSTKSZ are defined in <*signal.h*>.

After a successful call to one of the *exec* functions, there are no alternate signal stacks in the new process image.

In some implementations, a signal (whether or not indicated to execute on the alternate stack) shall always execute on the alternate stack if it is delivered while another signal is being caught using the alternate stack.

Use of this function by library threads that are not bound to kernel-scheduled entities results in undefined behavior.

RETURN VALUE

Upon successful completion, *sigaltstack*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *sigaltstack*() function shall fail if:

EINVAL
The *ss* argument is not a null pointer, and the *ss_flags* member pointed to by *ss* contains flags other than SS_DISABLE.

ENOMEM
The size of the alternate stack area is less than MINSIGSTKSZ.

EPERM
An attempt was made to modify an active stack.

The following sections are informative.

EXAMPLES

Allocating Memory for an Alternate Stack

The following example illustrates a method for allocating memory for an alternate stack.

```
#include <signal.h>
...
if ((sigstk.ss_sp = malloc(SIGSTKSZ)) == NULL)
    /* Error return. */
sigstk.ss_size = SIGSTKSZ;
sigstk.ss_flags = 0;
if (sigaltstack(&sigstk,(stack_t *)0) < 0)
    perror("sigaltstack");
```

APPLICATION USAGE

On some implementations, stack space is automatically extended as needed. On those implementations, automatic extension is typically not available for an alternate stack. If the stack overflows, the behavior is undefined.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *exec*, *sigaction*( ), *sigsetjmp*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigdelset — delete a signal from a signal set

SYNOPSIS

#include <signal.h>

int sigdelset(sigset_t **set*, int *signo*);

DESCRIPTION

The *sigdelset*() function deletes the individual signal specified by *signo* from the signal set pointed to by *set*.

Applications should call either *sigemptyset*() or *sigfillset*() at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigismember*(), *sigpending*(), *sigprocmask*(), *sigsuspend*(), *sigtimedwait*(), *sigwait*(), or *sigwaitinfo*(), the results are undefined.

RETURN VALUE

Upon successful completion, *sigdelset*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *sigdelset*() function may fail if:

EINVAL
The *signo* argument is not a valid signal number, or is an unsupported signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigemptyset — initialize and empty a signal set

SYNOPSIS

```
#include <signal.h>

int sigemptyset(sigset_t *set);
```

DESCRIPTION

The *sigemptyset*() function initializes the signal set pointed to by *set*, such that all signals defined in POSIX.1-2008 are excluded.

RETURN VALUE

Upon successful completion, *sigemptyset*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The implementation of the *sigemptyset*() (or *sigfillset*()) function could quite trivially clear (or set) all the bits in the signal set. Alternatively, it would be reasonable to initialize part of the structure, such as a version field, to permit binary-compatibility between releases where the size of the set varies. For such reasons, either *sigemptyset*() or *sigfillset*() must be called prior to any other use of the signal set, even if such use is read-only (for example, as an argument to *sigpending*()). This function is not intended for dynamic allocation.

The *sigfillset*() and *sigemptyset*() functions require that the resulting signal set include (or exclude) all the signals defined in this volume of POSIX.1-2017. Although it is outside the scope of this volume of POSIX.1-2017 to place this requirement on signals that are implemented as extensions, it is recommended that implementation-defined signals also be affected by these functions. However, there may be a good reason for a particular signal not to be affected. For example, blocking or ignoring an implementation-defined signal may have undesirable side-effects, whereas the default action for that signal is harmless. In such a case, it would be preferable for such a signal to be excluded from the signal set returned by *sigfillset*().

In early proposals there was no distinction between invalid and unsupported signals (the names of optional signals that were not supported by an implementation were not defined by that implementation). The **[EINVAL]** error was thus specified as a required error for invalid signals. With that distinction, it is not necessary to require implementations of these functions to determine whether an optional signal is actually supported, as that could have a significant performance impact for little value. The error could have been required for invalid signals and optional for unsupported signals, but this seemed unnecessarily complex. Thus, the error is optional in both cases.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigfillset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigfillset — initialize and fill a signal set

SYNOPSIS

```
#include <signal.h>

int sigfillset(sigset_t *set);
```

DESCRIPTION

The *sigfillset*() function shall initialize the signal set pointed to by *set*, such that all signals defined in this volume of POSIX.1-2017 are included.

RETURN VALUE

Upon successful completion, *sigfillset*() shall return 0; otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to *sigemptyset*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigismember*( ), *sigpending*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sighold, sigignore, sigpause, sigrelse, sigset — signal management

SYNOPSIS

```
#include <signal.h>

int sighold(int sig);
int sigignore(int sig);
int sigpause(int sig);
int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

DESCRIPTION

Use of any of these functions is unspecified in a multi-threaded process.

The *sighold*(), *sigignore*(), *sigpause*(), *sigrelse*(), and *sigset*() functions provide simplified signal management.

The *sigset*() function shall modify signal dispositions. The *sig* argument specifies the signal, which may be any signal except SIGKILL and SIGSTOP. The *disp* argument specifies the signal's disposition, which may be SIG_DFL, SIG_IGN, or the address of a signal handler. If *sigset*() is used, and *disp* is the address of a signal handler, the system shall add *sig* to the signal mask of the calling process before executing the signal handler; when the signal handler returns, the system shall restore the signal mask of the calling process to its state prior to the delivery of the signal. In addition, if *sigset*() is used, and *disp* is equal to SIG_HOLD, *sig* shall be added to the signal mask of the calling process and *sig*'s disposition shall remain unchanged. If *sigset*() is used, and *disp* is not equal to SIG_HOLD, *sig* shall be removed from the signal mask of the calling process.

The *sighold*() function shall add *sig* to the signal mask of the calling process.

The *sigrelse*() function shall remove *sig* from the signal mask of the calling process.

The *sigignore*() function shall set the disposition of *sig* to SIG_IGN.

The *sigpause*() function shall remove *sig* from the signal mask of the calling process and suspend the calling process until a signal is received. The *sigpause*() function shall restore the signal mask of the process to its original state before returning.

If the action for the SIGCHLD signal is set to SIG_IGN, child processes of the calling processes shall not be transformed into zombie processes when they terminate. If the calling process subsequently waits for its children, and the process has no unwaited-for children that were transformed into zombie processes, it shall block until all of its children terminate, and *wait*(), *waitid*(), and *waitpid*() shall fail and set *errno* to **[ECHILD]**.

RETURN VALUE

Upon successful completion, *sigset*() shall return SIG_HOLD if the signal had been blocked and the signal's previous disposition if it had not been blocked. Otherwise, SIG_ERR shall be returned and *errno* set to indicate the error.

The *sigpause*() function shall suspend execution of the thread until a signal is received, whereupon it shall return −1 and set *errno* to **[EINTR]**.

For all other functions, upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

These functions shall fail if:

EINVAL
: The *sig* argument is an illegal signal number.

The *sigset*() and *sigignore*() functions shall fail if:

EINVAL
: An attempt is made to catch a signal that cannot be caught, or to ignore a signal that cannot be ignored.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *sigaction*() function provides a more comprehensive and reliable mechanism for controlling signals; new applications should use the *sigaction*() function instead of the obsolescent *sigset*() function.

The *sighold*() function, in conjunction with *sigrelse*() or *sigpause*(), may be used to establish critical regions of code that require the delivery of a signal to be temporarily deferred. For broader portability, the *pthread_sigmask*() or *sigprocmask*() functions should be used instead of the obsolescent *sighold*() and *sigrelse*() functions.

For broader portability, the *sigsuspend*() function should be used instead of the obsolescent *sigpause*() function.

RATIONALE

Each of these historic functions has a direct analog in the other functions which are required to be per-thread and thread-safe (aside from *sigprocmask*(), which is replaced by *pthread_sigmask*()). The *sigset*() function can be implemented as a simple wrapper for *sigaction*(). The *sighold*() function is equivalent to *sigprocmask*() or *pthread_sigmask*() with SIG_BLOCK set. The *sigignore*() function is equivalent to *sigaction*() with SIG_IGN set. The *sigpause*() function is equivalent to *sigsuspend*(). The *sigrelse*() function is equivalent to *sigprocmask*() or *pthread_sigmask*() with SIG_UNBLOCK set.

FUTURE DIRECTIONS

These functions may be removed in a future version.

SEE ALSO

Section 2.4, *Signal Concepts*, *exec*, *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *signal*( ), *sigsuspend*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

siginterrupt — allow signals to interrupt functions

SYNOPSIS

```
#include <signal.h>

int siginterrupt(int sig, int flag);
```

DESCRIPTION

The *siginterrupt*() function shall change the restart behavior when a function is interrupted by the specified signal. The function *siginterrupt*(*sig*, *flag*) has an effect as if implemented as:

```
int siginterrupt(int sig, int flag) {
    int ret;
    struct sigaction act;

    (void) sigaction(sig, NULL, &act);
    if (flag)
        act.sa_flags &= ~SA_RESTART;
    else
        act.sa_flags |= SA_RESTART;
    ret = sigaction(sig, &act, NULL);
    return ret;
}
```

RETURN VALUE

Upon successful completion, *siginterrupt*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *siginterrupt*() function shall fail if:

EINVAL
The *sig* argument is not a valid signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *siginterrupt*() function supports programs written to historical system interfaces. Applications should use the *sigaction*() with the SA_RESTART flag instead of the obsolescent *siginterrupt*() function.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *sigaction*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigismember — test for a signal in a signal set

SYNOPSIS

#include <signal.h>

int sigismember(const sigset_t **set*, int *signo*);

DESCRIPTION

The *sigismember*() function shall test whether the signal specified by *signo* is a member of the set pointed to by *set*.

Applications should call either *sigemptyset*() or *sigfillset*() at least once for each object of type **sigset_t** prior to any other use of that object. If such an object is not initialized in this way, but is nonetheless supplied as an argument to any of *pthread_sigmask*(), *sigaction*(), *sigaddset*(), *sigdelset*(), *sigismember*(), *sigpending*(), *sigprocmask*(), *sigsuspend*(), *sigtimedwait*(), *sigwait*(), or *sigwaitinfo*(), the results are undefined.

RETURN VALUE

Upon successful completion, *sigismember*() shall return 1 if the specified signal is a member of the specified set, or 0 if it is not. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *sigismember*() function may fail if:

EINVAL

The *signo* argument is not a valid signal number, or is an unsupported signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pthread_sigmask*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigfillset*( ), *sigemptyset*( ), *sigpending*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

siglongjmp — non-local goto with signal handling

SYNOPSIS

#include <setjmp.h>

void siglongjmp(sigjmp_buf *env*, int *val*);

DESCRIPTION

The *siglongjmp*() function shall be equivalent to the *longjmp*() function, except as follows:

* References to *setjmp*() shall be equivalent to *sigsetjmp*().
* The *siglongjmp*() function shall restore the saved signal mask if and only if the *env* argument was initialized by a call to *sigsetjmp*() with a non-zero *savemask* argument.

RETURN VALUE

After *siglongjmp*() is completed, program execution shall continue as if the corresponding invocation of *sigsetjmp*() had just returned the value specified by *val*. The *siglongjmp*() function shall not cause *sigsetjmp*() to return 0; if *val* is 0, *sigsetjmp*() shall return the value 1.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*() or *longjmp*() and *sigsetjmp*() or *siglongjmp*() is only significant for programs which use *sigaction*(), *sigprocmask*(), or *sigsuspend*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

longjmp( ), *pthread_sigmask*( ), *setjmp*( ), *sigsetjmp*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<setjmp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

signal — signal management

SYNOPSIS

```
#include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *signal*() function chooses one of three ways in which receipt of the signal number *sig* is to be subsequently handled. If the value of *func* is SIG_DFL, default handling for that signal shall occur. If the value of *func* is SIG_IGN, the signal shall be ignored. Otherwise, the application shall ensure that *func* points to a function to be called when that signal occurs. An invocation of such a function because of a signal, or (recursively) of any further functions called by that invocation (other than functions in the standard library), is called a "signal handler".

When a signal occurs, and *func* points to a function, it is implementation-defined whether the equivalent of a:

```
signal(sig, SIG_DFL);
```

is executed or the implementation prevents some implementation-defined set of signals (at least including *sig*) from occurring until the current signal handling has completed. (If the value of *sig* is SIGILL, the implementation may alternatively define that no action is taken.) Next the equivalent of:

```
(*func)(sig);
```

is executed. If and when the function returns, if the value of *sig* was SIGFPE, SIGILL, or SIGSEGV or any other implementation-defined value corresponding to a computational exception, the behavior is undefined. Otherwise, the program shall resume execution at the point it was interrupted. The ISO C standard places a restriction on applications relating to the use of *raise*() from signal handlers. This restriction does not apply to POSIX applications, as POSIX.1-2008 requires *raise*() to be async-signal-safe (see *Section 2.4.3*, *Signal Actions*).

If the process is multi-threaded, or if the process is single-threaded and a signal handler is executed other than as the result of:

* The process calling *abort*(), *raise*(), *kill*(), *pthread_kill*(), or *sigqueue*() to generate a signal that is not blocked
* A pending signal being unblocked and being delivered before the call that unblocked it returns

the behavior is undefined if the signal handler refers to any object other than *errno* with static storage duration other than by assigning a value to an object declared as **volatile sig_atomic_t**, or if the signal handler calls any function defined in this standard other than one of the functions listed in *Section 2.4*, *Signal Concepts*.

At program start-up, the equivalent of:

```
signal(sig, SIG_IGN);
```

is executed for some signals, and the equivalent of:

```
signal(sig, SIG_DFL);
```

is executed for all other signals (see *exec*).

The *signal*() function shall not change the setting of *errno* if successful.

RETURN VALUE

If the request can be honored, *signal*() shall return the value of *func* for the most recent call to *signal*() for the specified signal *sig*. Otherwise, SIG_ERR shall be returned and a positive value shall be stored in *errno*.

ERRORS

The *signal*() function shall fail if:

EINVAL
The *sig* argument is not a valid signal number or an attempt is made to catch a signal that cannot be caught or ignore a signal that cannot be ignored.

The *signal*() function may fail if:

EINVAL
An attempt was made to set the action to SIG_DFL for a signal that cannot be caught or ignored (or both).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *sigaction*() function provides a more comprehensive and reliable mechanism for controlling signals; new applications should use *sigaction*() rather than *signal*().

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *exec*, *pause*( ), *raise*( ), *sigaction*( ), *sigsuspend*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

signbit — test sign

SYNOPSIS

#include <math.h>

int signbit(real-floating *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *signbit*() macro shall determine whether the sign of its argument value is negative. NaNs, zeros, and infinities have a sign bit.

RETURN VALUE

The *signbit*() macro shall return a non-zero value if and only if the sign of its argument value is negative.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fpclassify( ), *isfinite*( ), *isinf*( ), *isnan*( ), *isnormal*( )

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

signgam — log gamma function

SYNOPSIS

```
#include <math.h>

extern int signgam;
```

DESCRIPTION

Refer to *lgamma*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigpause — remove a signal from the signal mask and suspend the thread

SYNOPSIS

```
#include <signal.h>

int sigpause(int sig);
```

DESCRIPTION

Refer to *sighold*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigpending — examine pending signals

SYNOPSIS

```
#include <signal.h>

int sigpending(sigset_t *set);
```

DESCRIPTION

The *sigpending*() function shall store, in the location referenced by the *set* argument, the set of signals that are blocked from delivery to the calling thread and that are pending on the process or the calling thread.

RETURN VALUE

Upon successful completion, *sigpending*() shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *pthread_sigmask*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( ), *sigismember*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigprocmask — examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int sigprocmask(int how, const sigset_t *restrict set,
    sigset_t *restrict oset);
```

DESCRIPTION

Refer to *pthread_sigmask*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigqueue — queue a signal to a process

SYNOPSIS

```
#include <signal.h>

int sigqueue(pid_t pid, int signo, union sigval value);
```

DESCRIPTION

The *sigqueue*() function shall cause the signal specified by *signo* to be sent with the value specified by *value* to the process specified by *pid*. If *signo* is zero (the null signal), error checking is performed but no signal is actually sent. The null signal can be used to check the validity of *pid*.

The conditions required for a process to have permission to queue a signal to another process are the same as for the *kill*() function.

The *sigqueue*() function shall return immediately. If SA_SIGINFO is set for *signo* and if the resources were available to queue the signal, the signal shall be queued and sent to the receiving process. If SA_SIGINFO is not set for *signo*, then *signo* shall be sent at least once to the receiving process; it is unspecified whether *value* shall be sent to the receiving process as a result of this call.

If the value of *pid* causes *signo* to be generated for the sending process, and if *signo* is not blocked for the calling thread and if no other thread has *signo* unblocked or is waiting in a *sigwait*() function for *signo*, either *signo* or at least the pending, unblocked signal shall be delivered to the calling thread before the *sigqueue*() function returns. Should any multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected for delivery, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, the specified signal shall have been queued, and the *sigqueue*() function shall return a value of zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sigqueue*() function shall fail if:

EAGAIN
: No resources are available to queue the signal. The process has already queued {SIGQUEUE_MAX} signals that are still pending at the receiver(s), or a system-wide resource limit has been exceeded.

EINVAL
: The value of the *signo* argument is an invalid or unsupported signal number.

EPERM
: The process does not have appropriate privileges to send the signal to the receiving process.

ESRCH
: The process *pid* does not exist.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The *sigqueue*() function allows an application to queue a realtime signal to itself or to another process, specifying the application-defined value. This is common practice in realtime applications on existing realtime systems. It was felt that specifying another function in the *sig*... name space already carved out for signals was preferable to extending the interface to *kill*().

Such a function became necessary when the put/get event function of the message queues was removed. It should be noted that the *sigqueue*() function implies reduced performance in a security-conscious implementation as the access permissions between the sender and receiver have to be checked on each send when the *pid* is resolved into a target process. Such access checks were necessary only at message queue open in the previous interface.

The standard developers required that *sigqueue*() have the same semantics with respect to the null signal as *kill*(), and that the same permission checking be used. But because of the difficulty of implementing the ``broadcast'' semantic of *kill*() (for example, to process groups) and the interaction with resource allocation, this semantic was not adopted. The *sigqueue*() function queues a signal to a single process specified by the *pid* argument.

The *sigqueue*() function can fail if the system has insufficient resources to queue the signal. An explicit limit on the number of queued signals that a process could send was introduced. While the limit is ``per-sender'', this volume of POSIX.1-2017 does not specify that the resources be part of the state of the sender. This would require either that the sender be maintained after exit until all signals that it had sent to other processes were handled or that all such signals that had not yet been acted upon be removed from the queue(s) of the receivers. This volume of POSIX.1-2017 does not preclude this behavior, but an implementation that allocated queuing resources from a system-wide pool (with per-sender limits) and that leaves queued signals pending after the sender exits is also permitted.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.8.1, *Realtime Signals*

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigrelse, sigset — signal management

SYNOPSIS

```
#include <signal.h>

int sigrelse(int sig);
void (*sigset(int sig, void (*disp)(int)))(int);
```

DESCRIPTION

Refer to *sighold*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigsetjmp — set jump point for a non-local goto

SYNOPSIS

#include <setjmp.h>

int sigsetjmp(sigjmp_buf *env*, int *savemask*);

DESCRIPTION

The *sigsetjmp*() function shall be equivalent to the *setjmp*() function, except as follows:

- References to *setjmp*() are equivalent to *sigsetjmp*().
- References to *longjmp*() are equivalent to *siglongjmp*().
- If the value of the *savemask* argument is not 0, *sigsetjmp*() shall also save the current signal mask of the calling thread as part of the calling environment.

RETURN VALUE

If the return is from a successful direct invocation, *sigsetjmp*() shall return 0. If the return is from a call to *siglongjmp*(), *sigsetjmp*() shall return a non-zero value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The distinction between *setjmp*()/*longjmp*() and *sigsetjmp*()/*siglongjmp*() is only significant for programs which use *sigaction*(), *sigprocmask*(), or *sigsuspend*().

Note that since this function is defined in terms of *setjmp*(), if *savemask* is zero, it is unspecified whether the signal mask is saved.

RATIONALE

The ISO C standard specifies various restrictions on the usage of the *setjmp*() macro in order to permit implementors to recognize the name in the compiler and not implement an actual function. These same restrictions apply to the *sigsetjmp*() macro.

There are processors that cannot easily support these calls, but this was not considered a sufficient reason to exclude them.

4.2 BSD, 4.3 BSD, and XSI-conformant systems provide functions named *_setjmp*() and *_longjmp*() that, together with *setjmp*() and *longjmp*(), provide the same functionality as *sigsetjmp*() and *siglongjmp*(). On those systems, *setjmp*() and *longjmp*() save and restore signal masks, while *_setjmp*() and *_longjmp*() do not. On System V Release 3 and in corresponding issues of the SVID, *setjmp*() and *longjmp*() are explicitly defined not to save and restore signal masks. In order to permit existing practice in both cases, the relation of *setjmp*() and *longjmp*() to signal masks is not specified, and a new set of functions is defined instead.

The *longjmp*() and *siglongjmp*() functions operate as in the previous issue provided the matching *setjmp*() or *sigsetjmp*() has been performed in the same thread. Non-local jumps into contexts saved by other threads would be at best a questionable practice and were not considered worthy of standardization.

FUTURE DIRECTIONS

None.

SEE ALSO

pthread_sigmask( ), *siglongjmp*( ), *signal*( ), *sigsuspend*( )

The Base Definitions volume of POSIX.1-2017, **<setjmp.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigsuspend — wait for a signal

SYNOPSIS

```
#include <signal.h>

int sigsuspend(const sigset_t *sigmask);
```

DESCRIPTION

The *sigsuspend*() function shall replace the current signal mask of the calling thread with the set of signals pointed to by *sigmask* and then suspend the thread until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process. This shall not cause any other signals that may have been pending on the process to become pending on the thread.

If the action is to terminate the process then *sigsuspend*() shall never return. If the action is to execute a signal-catching function, then *sigsuspend*() shall return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the *sigsuspend*() call.

It is not possible to block signals that cannot be ignored. This is enforced by the system without causing an error to be indicated.

RETURN VALUE

Since *sigsuspend*() suspends thread execution indefinitely, there is no successful completion return value. If a return occurs, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *sigsuspend*() function shall fail if:

EINTR
A signal is caught by the calling process and control is returned from the signal-catching function.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Normally, at the beginning of a critical code section, a specified set of signals is blocked using the *sigprocmask*() function. When the thread has completed the critical section and needs to wait for the previously blocked signal(s), it pauses by calling *sigsuspend*() with the mask that was returned by the *sigprocmask*() call.

RATIONALE

Code which wants to avoid the ambiguity of the signal mask for thread cancellation handlers can install an additional cancellation handler which resets the signal mask to the expected value.

```
void cleanup(void *arg)
{
    sigset_t *ss = (sigset_t *) arg;
    pthread_sigmask(SIG_SETMASK, ss, NULL);
}

int call_sigsuspend(const sigset_t *mask)
{
    sigset_t oldmask;
    int result;
    pthread_sigmask(SIG_SETMASK, NULL, &oldmask);
```

```
        pthread_cleanup_push(cleanup, &oldmask);
        result = sigsuspend(sigmask);
        pthread_cleanup_pop(0);
        return result;
    }
```

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *pause*( ), *sigaction*( ), *sigaddset*( ), *sigdelset*( ), *sigemptyset*( ), *sigfillset*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigtimedwait, sigwaitinfo — wait for queued signals

SYNOPSIS

```
#include <signal.h>

int sigtimedwait(const sigset_t *restrict set,
   siginfo_t *restrict info,
   const struct timespec *restrict timeout);
int sigwaitinfo(const sigset_t *restrict set,
   siginfo_t *restrict info);
```

DESCRIPTION

The *sigtimedwait*() function shall be equivalent to *sigwaitinfo*() except that if none of the signals specified by *set* are pending, *sigtimedwait*() shall wait for the time interval specified in the **timespec** structure referenced by *timeout*. If the **timespec** structure pointed to by *timeout* is zero-valued and if none of the signals specified by *set* are pending, then *sigtimedwait*() shall return immediately with an error. If *timeout* is the null pointer, the behavior is unspecified. If the Monotonic Clock option is supported, the CLOCK_MONOTONIC clock shall be used to measure the time interval specified by the *timeout* argument.

The *sigwaitinfo*() function selects the pending signal from the set specified by *set*. Should any of multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified. If no signal in *set* is pending at the time of the call, the calling thread shall be suspended until one or more signals in *set* become pending or until it is interrupted by an unblocked, caught signal.

The *sigwaitinfo*() function shall be equivalent to the *sigwait*() function, except that the return value and the error reporting method are different (see RETURN VALUE), and that if the *info* argument is non-NULL, the selected signal number shall be stored in the *si_signo* member, and the cause of the signal shall be stored in the *si_code* member. If any value is queued to the selected signal, the first such queued value shall be dequeued and, if the *info* argument is non-NULL, the value shall be stored in the *si_value* member of *info*. The system resource used to queue the signal shall be released and returned to the system for other use. If no value is queued, the content of the *si_value* member is undefined. If no further signals are queued for the selected signal, the pending indication for that signal shall be reset.

RETURN VALUE

Upon successful completion (that is, one of the signals specified by *set* is pending or is generated) *sigwaitinfo*() and *sigtimedwait*() shall return the selected signal number. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sigtimedwait*() function shall fail if:

EAGAIN
No signal specified by *set* was generated within the specified timeout period.

The *sigtimedwait*() and *sigwaitinfo*() functions may fail if:

EINTR
The wait was interrupted by an unblocked, caught signal. It shall be documented in system documentation whether this error causes these functions to fail.

The *sigtimedwait*() function may also fail if:

EINVAL
The *timeout* argument specified a *tv_nsec* value less than zero or greater than or equal to 1 000 million.

An implementation should only check for this error if no signal is pending in *set* and it is necessary to wait.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *sigtimedwait*() function times out and returns an **[EAGAIN]** error. Application developers should note that this is inconsistent with other functions such as *pthread_cond_timedwait*() that return **[ETIMEDOUT]**.

Note that in order to ensure that generated signals are queued and signal values passed to *sigqueue*() are available in *si_value*, applications which use *sigwaitinfo*() or *sigtimedwait*() need to set the SA_SIGINFO flag for each signal in the set (see *Section 2.4*, *Signal Concepts*). This means setting each signal to be handled by a three-argument signal-catching function, even if the handler will never be called. It is not possible (portably) to set a signal handler to SIG_DFL while setting the SA_SIGINFO flag, because assigning to the *sa_handler* member of **struct sigaction** instead of the *sa_sigaction* member would result in undefined behavior, and SIG_DFL need not be assignment-compatible with *sa_sigaction*. Even if an assignment of SIG_DFL to *sa_sigaction* is accepted by the compiler, the implementation need not treat this value as special—it could just be taken as the address of a signal-catching function.

RATIONALE

Existing programming practice on realtime systems uses the ability to pause waiting for a selected set of events and handle the first event that occurs in-line instead of in a signal-handling function. This allows applications to be written in an event-directed style similar to a state machine. This style of programming is useful for largescale transaction processing in which the overall throughput of an application and the ability to clearly track states are more important than the ability to minimize the response time of individual event handling.

It is possible to construct a signal-waiting macro function out of the realtime signal function mechanism defined in this volume of POSIX.1-2017. However, such a macro has to include the definition of a generalized handler for all signals to be waited on. A significant portion of the overhead of handler processing can be avoided if the signal-waiting function is provided by the kernel. This volume of POSIX.1-2017 therefore provides two signal-waiting functions—one that waits indefinitely and one with a timeout—as part of the overall realtime signal function specification.

The specification of a function with a timeout allows an application to be written that can be broken out of a wait after a set period of time if no event has occurred. It was argued that setting a timer event before the wait and recognizing the timer event in the wait would also implement the same functionality, but at a lower performance level. Because of the performance degradation associated with the user-level specification of a timer event and the subsequent cancellation of that timer event after the wait completes for a valid event, and the complexity associated with handling potential race conditions associated with the user-level method, the separate function has been included.

Note that the semantics of the *sigwaitinfo*() function are nearly identical to that of the *sigwait*() function defined by this volume of POSIX.1-2017. The only difference is that *sigwaitinfo*() returns the queued signal value in the *value* argument. The return of the queued value is required so that applications can differentiate between multiple events queued to the same signal number.

The two distinct functions are being maintained because some implementations may choose to implement the POSIX Threads Extension functions and not implement the queued signals extensions. Note, though, that *sigwaitinfo*() does not return the queued value if the *value* argument is NULL, so the POSIX Threads Extension *sigwait*() function can be implemented as a macro on *sigwaitinfo*().

The *sigtimedwait*() function was separated from the *sigwaitinfo*() function to address concerns regarding the overloading of the *timeout* pointer to indicate indefinite wait (no timeout), timed wait, and immediate return, and concerns regarding consistency with other functions where the conditional and timed waits were separate functions from the pure blocking function. The semantics of *sigtimedwait*() are specified such that *sigwaitinfo*() could be implemented as a macro with a null pointer for *timeout*.

The *sigwait* functions provide a synchronous mechanism for threads to wait for asynchronously-generated signals. One important question was how many threads that are suspended in a call to a *sigwait*() function for a signal should return from the call when the signal is sent. Four choices were considered:

1. Return an error for multiple simultaneous calls to *sigwait* functions for the same signal.
2. One or more threads return.
3. All waiting threads return.
4. Exactly one thread returns.

Prohibiting multiple calls to *sigwait*() for the same signal was felt to be overly restrictive. The ''one or more'' behavior made implementation of conforming packages easy at the expense of forcing POSIX threads clients to protect against multiple simultaneous calls to *sigwait*() in application code in order to achieve predictable behavior. There was concern that the ''all waiting threads'' behavior would result in ''signal broadcast storms'', consuming excessive CPU resources by replicating the signals in the general case. Furthermore, no convincing examples could be presented that delivery to all was either simpler or more powerful than delivery to one.

Thus, the consensus was that exactly one thread that was suspended in a call to a *sigwait* function for a signal should return when that signal occurs. This is not an onerous restriction as:

* A multi-way signal wait can be built from the single-way wait.
* Signals should only be handled by application-level code, as library routines cannot guess what the application wants to do with signals generated for the entire process.
* Applications can thus arrange for a single thread to wait for any given signal and call any needed routines upon its arrival.

In an application that is using signals for interprocess communication, signal processing is typically done in one place. Alternatively, if the signal is being caught so that process cleanup can be done, the signal handler thread can call separate process cleanup routines for each portion of the application. Since the application main line started each portion of the application, it is at the right abstraction level to tell each portion of the application to clean up.

Certainly, there exist programming styles where it is logical to consider waiting for a single signal in multiple threads. A simple *sigwait_multiple*() routine can be constructed to achieve this goal. A possible implementation would be to have each *sigwait_multiple*() caller registered as having expressed interest in a set of signals. The caller then waits on a thread-specific condition variable. A single server thread calls a *sigwait*() function on the union of all registered signals. When the *sigwait*() function returns, the appropriate state is set and condition variables are broadcast. New *sigwait_multiple*() callers may cause the pending *sigwait*() call to be canceled and reissued in order to update the set of signals being waited for.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *Section 2.8.1*, *Realtime Signals*, *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigwait*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigwait — wait for queued signals

SYNOPSIS

#include <signal.h>

int sigwait(const sigset_t *restrict *set*, int *restrict *sig*);

DESCRIPTION

The *sigwait*() function shall select a pending signal from *set*, atomically clear it from the system's set of pending signals, and return that signal number in the location referenced by *sig*. If prior to the call to *sigwait*() there are multiple pending instances of a single signal number, it is implementation-defined whether upon successful return there are any remaining pending signals for that signal number. If the implementation supports queued signals and there are multiple signals queued for the signal number selected, the first such queued signal shall cause a return from *sigwait*() and the remainder shall remain queued. If no signal in *set* is pending at the time of the call, the thread shall be suspended until one or more becomes pending. The signals defined by *set* shall have been blocked at the time of the call to *sigwait*(); otherwise, the behavior is undefined. The effect of *sigwait*() on the signal actions for the signals in *set* is unspecified.

If more than one thread is using *sigwait*() to wait for the same signal, no more than one of these threads shall return from *sigwait*() with the signal number. If more than a single thread is blocked in *sigwait*() for a signal when that signal is generated for the process, it is unspecified which of the waiting threads returns from *sigwait*(). If the signal is generated for a specific thread, as by *pthread_kill*(), only that thread shall return.

Should any of the multiple pending signals in the range SIGRTMIN to SIGRTMAX be selected, it shall be the lowest numbered one. The selection order between realtime and non-realtime signals, or between multiple pending non-realtime signals, is unspecified.

RETURN VALUE

Upon successful completion, *sigwait*() shall store the signal number of the received signal at the location referenced by *sig* and return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *sigwait*() function may fail if:

EINVAL
The *set* argument contains an invalid or unsupported signal number.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

To provide a convenient way for a thread to wait for a signal, this volume of POSIX.1-2017 provides the *sigwait*() function. For most cases where a thread has to wait for a signal, the *sigwait*() function should be quite convenient, efficient, and adequate.

However, requests were made for a lower-level primitive than *sigwait*() and for semaphores that could be used by threads. After some consideration, threads were allowed to use semaphores and *sem_post*() was defined to be async-signal-safe.

In summary, when it is necessary for code run in response to an asynchronous signal to notify a thread, *sigwait*() should be used to handle the signal. Alternatively, if the implementation provides semaphores, they also can be used, either following *sigwait*() or from within a signal handling routine previously registered with *sigaction*().

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4, *Signal Concepts*, *Section 2.8.1*, *Realtime Signals*, *pause*( ), *pthread_sigmask*( ), *sigaction*( ), *sigpending*( ), *sigsuspend*( ), *sigtimedwait*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sigwaitinfo — wait for queued signals

SYNOPSIS

#include <signal.h>

int sigwaitinfo(const sigset_t *restrict *set*, siginfo_t *restrict *info*);

DESCRIPTION

Refer to *sigtimedwait*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sin, sinf, sinl — sine function

SYNOPSIS

```
#include <math.h>

double sin(double x);
float sinf(float x);
long double sinl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the sine of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the sine of *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *sin*(), *sinf*(), and *sinl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is ±Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The *x* argument is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

These functions may fail if:

Range Error The value of *x* is subnormal

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Taking the Sine of a 45-Degree Angle

```
#include <math.h>
...
```

```
double radians = 45.0 * M_PI / 180;
double result;
...
result = sin(radians);
```

APPLICATION USAGE

These functions may lose accuracy when their argument is near a multiple of π or is far from 0.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asin( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sinh, sinhf, sinhl — hyperbolic sine functions

SYNOPSIS

#include <math.h>

double sinh(double *x*);
float sinhf(float *x*);
long double sinhl(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the hyperbolic sine of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic sine of *x*.

If the result would cause an overflow, a range error shall occur and ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL (with the same sign as *x*) shall be returned as appropriate for the type of the function.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *sinh*(), *sinhf*(), and *sinhl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions shall fail if:

Range Error The result would cause an overflow.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The value *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

asinh( ), *cosh*( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tanh*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sinl — sine function

SYNOPSIS

```
#include <math.h>

long double sinl(long double x);
```

DESCRIPTION

Refer to *sin*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sleep — suspend execution for an interval of time

SYNOPSIS

#include <unistd.h>

unsigned sleep(unsigned *seconds*);

DESCRIPTION

The *sleep*() function shall cause the calling thread to be suspended from execution until either the number of realtime seconds specified by the argument *seconds* has elapsed or a signal is delivered to the calling thread and its action is to invoke a signal-catching function or to terminate the process. The suspension time may be longer than requested due to the scheduling of other activity by the system.

In single-threaded programs, *sleep*() may make use of SIGALRM. In multi-threaded programs, *sleep*() shall not make use of SIGALRM and the remainder of this DESCRIPTION does not apply.

If a SIGALRM signal is generated for the calling process during execution of *sleep*() and if the SIGALRM signal is being ignored or blocked from delivery, it is unspecified whether *sleep*() returns when the SIGALRM signal is scheduled. If the signal is being blocked, it is also unspecified whether it remains pending after *sleep*() returns or it is discarded.

If a SIGALRM signal is generated for the calling process during execution of *sleep*(), except as a result of a prior call to *alarm*(), and if the SIGALRM signal is not being ignored or blocked from delivery, it is unspecified whether that signal has any effect other than causing *sleep*() to return.

If a signal-catching function interrupts *sleep*() and examines or changes either the time a SIGALRM is scheduled to be generated, the action associated with the SIGALRM signal, or whether the SIGALRM signal is blocked from delivery, the results are unspecified.

If a signal-catching function interrupts *sleep*() and calls *siglongjmp*() or *longjmp*() to restore an environment saved prior to the *sleep*() call, the action associated with the SIGALRM signal and the time at which a SIGALRM signal is scheduled to be generated are unspecified. It is also unspecified whether the SIGALRM signal is blocked, unless the signal mask of the process is restored as part of the environment.

Interactions between *sleep*() and *setitimer*() are unspecified.

RETURN VALUE

If *sleep*() returns because the requested time has elapsed, the value returned shall be 0. If *sleep*() returns due to delivery of a signal, the return value shall be the "unslept" amount (the requested time minus the time actually slept) in seconds.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

There are two general approaches to the implementation of the *sleep*() function. One is to use the *alarm*() function to schedule a SIGALRM signal and then suspend the calling thread waiting for that signal. The other is to implement an independent facility. This volume of POSIX.1-2017 permits either approach in single-threaded programs, but the simple alarm/suspend implementation is not appropriate for multi-threaded programs.

In order to comply with the requirement that no primitive shall change a process attribute unless explicitly described by this volume of POSIX.1-2017, an implementation using SIGALRM must carefully take into account any SIGALRM signal scheduled by previous *alarm*() calls, the action previously established for SIGALRM, and whether SIGALRM was blocked. If a SIGALRM has been scheduled before the *sleep*() would ordinarily complete, the *sleep*() must be shortened to that time and a SIGALRM generated (possibly simulated by direct invocation of the signal-catching function) before *sleep*() returns. If a SIGALRM has been scheduled after the *sleep*() would ordinarily complete, it must be rescheduled for the same time before *sleep*() returns. The action and blocking for SIGALRM must be saved and restored.

Historical implementations often implement the SIGALRM-based version using *alarm*() and *pause*(). One such implementation is prone to infinite hangups, as described in *pause*( ). Another such implementation uses the C-language *setjmp*() and *longjmp*() functions to avoid that window. That implementation introduces a different problem: when the SIGALRM signal interrupts a signal-catching function installed by the user to catch a different signal, the *longjmp*() aborts that signal-catching function. An implementation based on *sigprocmask*(), *alarm*(), and *sigsuspend*() can avoid these problems.

Despite all reasonable care, there are several very subtle, but detectable and unavoidable, differences between the two types of implementations. These are the cases mentioned in this volume of POSIX.1-2017 where some other activity relating to SIGALRM takes place, and the results are stated to be unspecified. All of these cases are sufficiently unusual as not to be of concern to most applications.

See also the discussion of the term *realtime* in *alarm*( ).

Since *sleep*() can be implemented using *alarm*(), the discussion about alarms occurring early under *alarm*() applies to *sleep*() as well.

Application developers should note that the type of the argument *seconds* and the return value of *sleep*() is **unsigned**. That means that a Strictly Conforming POSIX System Interfaces Application cannot pass a value greater than the minimum guaranteed value for {UINT_MAX}, which the ISO C standard sets as 65 535, and any application passing a larger value is restricting its portability. A different type was considered, but historical implementations, including those with a 16-bit **int** type, consistently use either **unsigned** or **int**.

Scheduling delays may cause the process to return from the *sleep*() function significantly after the requested time. In such cases, the return value should be set to zero, since the formula (requested time minus the time actually spent) yields a negative number and *sleep*() returns an **unsigned**.

FUTURE DIRECTIONS

A future version of this standard may require that *sleep*() does not make use of SIGALRM in all programs, not just multi-threaded programs.

SEE ALSO

alarm( ), *getitimer*( ), *nanosleep*( ), *pause*( ), *sigaction*( ), *sigsetjmp*( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

snprintf — print formatted output

SYNOPSIS

```
#include <stdio.h>

int snprintf(char *restrict s, size_t n,
   const char *restrict format, ...);
```

DESCRIPTION

Refer to *fprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sockatmark — determine whether a socket is at the out-of-band mark

SYNOPSIS

```
#include <sys/socket.h>

int sockatmark(int s);
```

DESCRIPTION

The *sockatmark*() function shall determine whether the socket specified by the descriptor *s* is at the out-of-band data mark (see *Section 2.10.12*, *Socket Out-of-Band Data State*). If the protocol for the socket supports out-of-band data by marking the stream with an out-of-band data mark, the *sockatmark*() function shall return 1 when all data preceding the mark has been read and the out-of-band data mark is the first element in the receive queue. The *sockatmark*() function shall not remove the mark from the stream.

RETURN VALUE

Upon successful completion, the *sockatmark*() function shall return a value indicating whether the socket is at an out-of-band data mark. If the protocol has marked the data stream and all data preceding the mark has been read, the return value shall be 1; if there is no mark, or if data precedes the mark in the receive queue, the *sockatmark*() function shall return 0. Otherwise, it shall return a value of −1 and set *errno* to indicate the error.

ERRORS

The *sockatmark*() function shall fail if:

EBADF
The *s* argument is not a valid file descriptor.

ENOTTY
The file associated with the *s* argument is not a socket.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The use of this function between receive operations allows an application to determine which received data precedes the out-of-band data and which follows the out-of-band data.

There is an inherent race condition in the use of this function. On an empty receive queue, the current read of the location might well be at the ''mark'', but the system has no way of knowing that the next data segment that will arrive from the network will carry the mark, and *sockatmark*() will return false, and the next read operation will silently consume the mark.

Hence, this function can only be used reliably when the application already knows that the out-of-band data has been seen by the system or that it is known that there is data waiting to be read at the socket (via SIGURG or *select*()). See *Section 2.10.11*, *Socket Receive Queue*, *Section 2.10.12*, *Socket Out-of-Band Data State*, *Section 2.10.14*, *Signals*, and *pselect*() for details.

RATIONALE

The *sockatmark*() function replaces the historical SIOCATMARK command to *ioctl*() which implemented the same functionality on many implementations. Using a wrapper function follows the adopted conventions to avoid specifying commands to the *ioctl*() function, other than those now included to support XSI STREAMS. The *sockatmark*() function could be implemented as follows:

```
#include <sys/ioctl.h>
```

```
int sockatmark(int s)
{
    int val;
    if (ioctl(s,SIOCATMARK,&val)==-1)
        return(-1);
    return(val);
}
```

The use of **[ENOTTY]** to indicate an incorrect descriptor type matches the historical behavior of SIOCATMARK.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.10.12, *Socket Out-of-Band Data State*, *pselect*( ), *recv*( ), *recvmsg*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

socket — create an endpoint for communication

SYNOPSIS

```
#include <sys/socket.h>

int socket(int domain, int type, int protocol);
```

DESCRIPTION

The *socket*() function shall create an unbound socket in a communications domain, and return a file descriptor that can be used in later function calls that operate on sockets. The file descriptor shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*.

The *socket*() function takes the following arguments:

domain	Specifies the communications domain in which a socket is to be created.
type	Specifies the type of socket to be created.
protocol	Specifies a particular protocol to be used with the socket. Specifying a *protocol* of 0 causes *socket*() to use an unspecified default protocol appropriate for the requested socket type.

The *domain* argument specifies the address family used in the communications domain. The address families supported by the system are implementation-defined.

Symbolic constants that can be used for the domain argument are defined in the *<sys/socket.h>* header.

The *type* argument specifies the socket type, which determines the semantics of communication over the socket. The following socket types are defined; implementations may specify additional socket types:

SOCK_STREAM
: Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM
: Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_SEQPACKET
: Provides sequenced, reliable, bidirectional, connection-mode transmission paths for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address family. If the *protocol* argument is zero, the default protocol for this address family and type shall be used. The protocols supported by the system are implementation-defined.

The process may need to have appropriate privileges to use the *socket*() function or to create some sockets.

RETURN VALUE

Upon successful completion, *socket*() shall return a non-negative integer, the socket file descriptor. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *socket*() function shall fail if:

EAFNOSUPPORT
: The implementation does not support the specified address family.

EMFILE
All file descriptors available to the process are currently open.

ENFILE
No more file descriptors are available for the system.

EPROTONOSUPPORT
The protocol is not supported by the address family, or the protocol is not supported by the implementation.

EPROTOTYPE
The socket type is not supported by the protocol.

The *socket*() function may fail if:

EACCES
The process does not have appropriate privileges.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

The application can determine whether an address family is supported by trying to create a socket with *domain* set to the protocol in question.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *accept*( ), *bind*( ), *connect*( ), *getsockname*( ), *getsockopt*( ), *listen*( ), *recv*( ), *recvfrom*( ), *recvmsg*( ), *send*( ), *sendmsg*( ), *setsockopt*( ), *shutdown*( ), *socketpair*( )

The Base Definitions volume of POSIX.1-2017, **<netinet_in.h>**, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

socketpair — create a pair of connected sockets

SYNOPSIS

```
#include <sys/socket.h>

int socketpair(int domain, int type, int protocol,
   int socket_vector[2]);
```

DESCRIPTION

The *socketpair*() function shall create an unbound pair of connected sockets in a specified *domain*, of a specified *type*, under the protocol optionally specified by the *protocol* argument. The two sockets shall be identical. The file descriptors used in referencing the created sockets shall be returned in *socket_vector*[0] and *socket_vector*[1]. The file descriptors shall be allocated as described in *Section 2.14*, *File Descriptor Allocation*.

The *socketpair*() function takes the following arguments:

domain	Specifies the communications domain in which the sockets are to be created.
type	Specifies the type of sockets to be created.
protocol	Specifies a particular protocol to be used with the sockets. Specifying a *protocol* of 0 causes *socketpair*() to use an unspecified default protocol appropriate for the requested socket type.
socket_vector	Specifies a 2-integer array to hold the file descriptors of the created socket pair.

The *type* argument specifies the socket type, which determines the semantics of communications over the socket. The following socket types are defined; implementations may specify additional socket types:

SOCK_STREAM
: Provides sequenced, reliable, bidirectional, connection-mode byte streams, and may provide a transmission mechanism for out-of-band data.

SOCK_DGRAM
: Provides datagrams, which are connectionless-mode, unreliable messages of fixed maximum length.

SOCK_SEQPACKET
: Provides sequenced, reliable, bidirectional, connection-mode transmission paths for records. A record can be sent using one or more output operations and received using one or more input operations, but a single operation never transfers part of more than one record. Record boundaries are visible to the receiver via the MSG_EOR flag.

If the *protocol* argument is non-zero, it shall specify a protocol that is supported by the address family. If the *protocol* argument is zero, the default protocol for this address family and type shall be used. The protocols supported by the system are implementation-defined.

The process may need to have appropriate privileges to use the *socketpair*() function or to create some sockets.

RETURN VALUE

Upon successful completion, this function shall return 0; otherwise, −1 shall be returned and *errno* set to indicate the error, no file descriptors shall be allocated, and the contents of *socket_vector* shall be left unmodified.

ERRORS

The *socketpair*() function shall fail if:

EAFNOSUPPORT
The implementation does not support the specified address family.

EMFILE
All, or all but one, of the file descriptors available to the process are currently open.

ENFILE
No more file descriptors are available for the system.

EOPNOTSUPP
The specified protocol does not permit creation of socket pairs.

EPROTONOSUPPORT
The protocol is not supported by the address family, or the protocol is not supported by the implementation.

EPROTOTYPE
The socket type is not supported by the protocol.

The *socketpair*() function may fail if:

EACCES
The process does not have appropriate privileges.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENOMEM
Insufficient memory was available to fulfill the request.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The documentation for specific address families specifies which protocols each address family supports. The documentation for specific protocols specifies which socket types each protocol supports.

The *socketpair*() function is used primarily with UNIX domain sockets and need not be supported for other domains.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.14, *File Descriptor Allocation*, *socket*( )

The Base Definitions volume of POSIX.1-2017, **<sys_socket.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sprintf — print formatted output

SYNOPSIS

#include <stdio.h>

int sprintf(char *restrict *s*, const char *restrict *format*, ...);

DESCRIPTION

Refer to *fprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

sqrt, sqrtf, sqrtl — square root function

SYNOPSIS

```
#include <math.h>

double sqrt(double x);
float sqrtf(float x);
long double sqrtl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the square root of their argument *x*, $sqrt {x}$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the square root of *x*.

For finite values of *x* < −0, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or +Inf, *x* shall be returned.

If *x* is −Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The finite value of *x* is < −0, or *x* is −Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Taking the Square Root of 9.0

```
#include <math.h>
...
double x = 9.0;
double result;
...
result = sqrt(x);
```

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

srand — pseudo-random number generator

SYNOPSIS

#include <stdlib.h>

void srand(unsigned *seed*);

DESCRIPTION

Refer to *rand*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

srand48 — seed the uniformly distributed double-precision pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>

void srand48(long seedval);
```

DESCRIPTION

Refer to *drand48*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

srandom — seed pseudo-random number generator

SYNOPSIS

```
#include <stdlib.h>

void srandom(unsigned seed);
```

DESCRIPTION

Refer to *initstate*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sscanf — convert formatted input

SYNOPSIS

#include <stdio.h>

int sscanf(const char *restrict *s*, const char *restrict *format*, ...);

DESCRIPTION

Refer to *fscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stat — get file status

SYNOPSIS

#include <sys/stat.h>

int stat(const char *restrict *path*, struct stat *restrict *buf*);

DESCRIPTION

Refer to *fstatat*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

statvfs — get file system information

SYNOPSIS

#include <sys/statvfs.h>

int statvfs(const char *restrict *path*, struct statvfs *restrict *buf*);

DESCRIPTION

Refer to *fstatvfs*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stderr, stdin, stdout — standard I/O streams

SYNOPSIS

```
#include <stdio.h>

extern FILE *stderr, *stdin, *stdout;
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type **FILE**. The *fopen*() function shall create certain descriptive data for a stream and return a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the *<stdio.h>* header and associated with the standard open files.

At program start-up, three streams shall be predefined and need not be opened explicitly: *standard input* (for reading conventional input), *standard output* (for writing conventional output), and *standard error* (for writing diagnostic output). When opened, the standard error stream is not fully buffered; the standard input and standard output streams are fully buffered if and only if the stream can be determined not to refer to an interactive device.

The following symbolic values in *<unistd.h>* define the file descriptors that shall be associated with the C-language *stdin*, *stdout*, and *stderr* when the application is started:

STDIN_FILENO
Standard input value, *stdin*. Its value is 0.

STDOUT_FILENO
Standard output value, *stdout*. Its value is 1.

STDERR_FILENO
Standard error value, *stderr*. Its value is 2.

The *stderr* stream is expected to be open for reading and writing.

RETURN VALUE

None.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fclose( ), *feof*( ), *ferror*( ), *fileno*( ), *fopen*( ), *fprintf*( ), *fread*( ), *fscanf*( ), *fseek*( ), *getc*( ), *gets*( ), *popen*( ), *putc*( ), *puts*( ), *read*( ), *setbuf*( ), *setvbuf*( ), *tmpfile*( ), *ungetc*( ), *vfprintf*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stpcpy — copy a string and return a pointer to the end of the result

SYNOPSIS

#include <string.h>

char *stpcpy(char *restrict *s1*, const char *restrict *s2*);

DESCRIPTION

Refer to *strcpy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stpncpy — copy fixed length string, returning a pointer to the array end

SYNOPSIS

#include <string.h>

char *stpncpy(char *restrict *s1*, const char *restrict *s2*, size_t *size*);

DESCRIPTION

Refer to *strncpy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strcasecmp, strcasecmp_l, strncasecmp, strncasecmp_l — case-insensitive string comparisons

SYNOPSIS

```
#include <strings.h>

int strcasecmp(const char *s1, const char *s2);
int strcasecmp_l(const char *s1, const char *s2,
    locale_t locale);
int strncasecmp(const char *s1, const char *s2, size_t n);
int strncasecmp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

DESCRIPTION

The *strcasecmp*() and *strcasecmp_l*() functions shall compare, while ignoring differences in case, the string pointed to by *s1* to the string pointed to by *s2*. The *strncasecmp*() and *strncasecmp_l*() functions shall compare, while ignoring differences in case, not more than *n* bytes from the string pointed to by *s1* to the string pointed to by *s2*.

The *strcasecmp*() and *strncasecmp*() functions use the current locale to determine the case of the characters.

The *strcasecmp_l*() and *strncasecmp_l*() functions use the locale represented by *locale* to determine the case of the characters.

When the *LC_CTYPE* category of the locale being used is from the POSIX locale, these functions shall behave as if the strings had been converted to lowercase and then a byte comparison performed. Otherwise, the results are unspecified.

The behavior is undefined if the *locale* argument to *strcasecmp_l*() or *strncasecmp_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, *strcasecmp*() and *strcasecmp_l*() shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is, ignoring case, greater than, equal to, or less than the string pointed to by *s2*, respectively.

Upon successful completion, *strncasecmp*() and *strncasecmp_l*() shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is, ignoring case, greater than, equal to, or less than the possibly null-terminated array pointed to by *s2*, respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscasecmp( )

The Base Definitions volume of POSIX.1-2017, **<strings.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strcat — concatenate two strings

SYNOPSIS

#include <string.h>

char *strcat(char *restrict *s1*, const char *restrict *s2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strcat*() function shall append a copy of the string pointed to by *s2* (including the terminating NUL character) to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strcat*() function shall return *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncat( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strchr — string scanning operation

SYNOPSIS

#include <string.h>

char *strchr(const char **s*, int *c*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strchr*() function shall locate the first occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating NUL character is considered to be part of the string.

RETURN VALUE

Upon completion, *strchr*() shall return a pointer to the byte, or a null pointer if the byte was not found.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strrchr( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strcmp — compare two strings

SYNOPSIS

#include <string.h>

int strcmp(const char **s1*, const char **s2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strcmp*() function shall compare the string pointed to by *s1* to the string pointed to by *s2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE

Upon completion, *strcmp*() shall return an integer greater than, equal to, or less than 0, if the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2*, respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Checking a Password Entry

The following example compares the information read from standard input to the value of the name of the user entry. If the *strcmp*() function returns 0 (indicating a match), a further check will be made to see if the user entered the proper old password. The *crypt*() function shall encrypt the old password entered by the user, using the value of the encrypted password in the **passwd** structure as the salt. If this value matches the value of the encrypted **passwd** in the structure, the entered password *oldpasswd* is the correct user's password. Finally, the program encrypts the new password so that it can store the information in the **passwd** structure.

```
#include <string.h>
#include <unistd.h>
#include <stdio.h>
...
int valid_change;
struct passwd *p;
char user[100];
char oldpasswd[100];
char newpasswd[100];
char savepasswd[100];
...
if (strcmp(p->pw_name, user) == 0) {
    if (strcmp(p->pw_passwd, crypt(oldpasswd, p->pw_passwd)) == 0) {
        strcpy(savepasswd, crypt(newpasswd, user));
        p->pw_passwd = savepasswd;
        valid_change = 1;
    }
```

```
        else {
            fprintf(stderr, "Old password is not valid\n");
        }
    }
    ...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncmp( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strcoll, strcoll_l — string comparison using collating information

SYNOPSIS

```
#include <string.h>

int strcoll(const char *s1, const char *s2);
int strcoll_l(const char *s1, const char *s2,
    locale_t locale);
```

DESCRIPTION

For *strcoll*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strcoll*() and *strcoll_l*() functions shall compare the string pointed to by *s1* to the string pointed to by *s2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale, or of the locale represented by *locale*, respectively.

The *strcoll*() and *strcoll_l*() functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strcoll*(), or *strcoll_l*() then check *errno*.

The behavior is undefined if the *locale* argument to *strcoll_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *strcoll*() shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the current locale. On error, *strcoll*() may set *errno*, but no return value is reserved to indicate an error.

Upon successful completion, *strcoll_l*() shall return an integer greater than, equal to, or less than 0, according to whether the string pointed to by *s1* is greater than, equal to, or less than the string pointed to by *s2* when both are interpreted as appropriate to the locale represented by *locale*. On error, *strcoll_l*() may set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

EINVAL
The *s1* or *s2* arguments contain characters outside the domain of the collating sequence.

The following sections are informative.

EXAMPLES

Comparing Nodes

The following example uses an application-defined function, *node_compare*(), to compare two nodes based on an alphabetical ordering of the *string* field.

```
#include <string.h>
...
struct node { /* These are stored in the table. */
    char *string;
    int length;
};
```

```
...
int node_compare(const void *node1, const void *node2)
{
    return strcoll(((const struct node *)node1)->string,
        ((const struct node *)node2)->string);
}
...
```

APPLICATION USAGE

The *strxfrm*() and *strcmp*() functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

alphasort( ), *strcmp*( ), *strxfrm*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stpcpy, strcpy — copy a string and return a pointer to the end of the result

SYNOPSIS

```
#include <string.h>

char *stpcpy(char *restrict s1, const char *restrict s2);
char *strcpy(char *restrict s1, const char *restrict s2);
```

DESCRIPTION

For *strcpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *stpcpy*() and *strcpy*() functions shall copy the string pointed to by *s2* (including the terminating NUL character) into the array pointed to by *s1*.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *stpcpy*() function shall return a pointer to the terminating NUL character copied into the *s1* buffer.

The *strcpy*() function shall return *s1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Construction of a Multi-Part Message in a Single Buffer

```
#include <string.h>
#include <stdio.h>

int
main (void)
{
    char buffer [10];
    char *name = buffer;

    name = stpcpy (stpcpy (stpcpy (name, "ice"),"-"), "cream");
    puts (buffer);
    return 0;
}
```

Initializing a String

The following example copies the string **"----------"** into the *permstring* variable.

```
#include <string.h>
...
static char permstring[11];
...
strcpy(permstring, "----------");
...
```

Storing a Key and Data

The following example allocates space for a key using *malloc*() then uses *strcpy*() to place the key there. Then it allocates space for data using *malloc*(), and uses *strcpy*() to place data there. (The user-defined function *dbfree*() frees memory previously allocated to an array of type **struct element ***.)

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
...
/* Structure used to read data and store it. */
struct element {
    char *key;
    char *data;
};

struct element *tbl, *curtbl;
char *key, *data;
int count;
...
void dbfree(struct element *, int);
...
if ((curtbl->key = malloc(strlen(key) + 1)) == NULL) {
    perror("malloc"); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->key, key);

if ((curtbl->data = malloc(strlen(data) + 1)) == NULL) {
    perror("malloc"); free(curtbl->key); dbfree(tbl, count); return NULL;
}
strcpy(curtbl->data, data);
...
```

APPLICATION USAGE

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

This version is aligned with the ISO C standard; this does not affect compatibility with XPG3 applications. Reliable error detection by this function was never guaranteed.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy( ), *wcscpy*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strcspn — get the length of a complementary substring

SYNOPSIS

```
#include <string.h>

size_t strcspn(const char *s1, const char *s2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strcspn*() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by *s1* which consists entirely of bytes *not* from the string pointed to by *s2*.

RETURN VALUE

The *strcspn*() function shall return the length of the computed segment of the string pointed to by *s1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strspn( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strdup, strndup — duplicate a specific number of bytes from a string

SYNOPSIS

```
#include <string.h>

char *strdup(const char *s);
char *strndup(const char *s, size_t size);
```

DESCRIPTION

The *strdup*() function shall return a pointer to a new string, which is a duplicate of the string pointed to by *s*. The returned pointer can be passed to *free*(). A null pointer is returned if the new string cannot be created.

The *strndup*() function shall be equivalent to the *strdup*() function, duplicating the provided *s* in a new block of memory allocated as if by using *malloc*(), with the exception being that *strndup*() copies at most *size* plus one bytes into the newly allocated memory, terminating the new string with a NUL character. If the length of *s* is larger than *size*, only *size* bytes shall be duplicated. If *size* is larger than the length of *s*, all bytes in *s* shall be copied into the new memory buffer, including the terminating NUL character. The newly created string shall always be properly terminated.

RETURN VALUE

The *strdup*() function shall return a pointer to a new string on success. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

Upon successful completion, the *strndup*() function shall return a pointer to the newly allocated memory containing the duplicated string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

ENOMEM
Storage space available is insufficient.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *strdup*() and *strndup*(), this is the return value.

Implementations are free to *malloc*() a buffer containing either (*size* + 1) bytes or (*strnlen*(*s*, *size*) + 1) bytes. Applications should not assume that *strndup*() will allocate (*size* + 1) bytes when *strlen*(*s*) is smaller than *size*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *wcsdup*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strerror, strerror_l, strerror_r — get error message string

SYNOPSIS

#include <string.h>

char *strerror(int *errnum*);
char *strerror_l(int *errnum*, locale_t *locale*);
int strerror_r(int *errnum*, char **strerrbuf*, size_t *buflen*);

DESCRIPTION

For *strerror*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strerror*() function shall map the error number in *errnum* to a locale-dependent error message string and shall return a pointer to it. Typically, the values for *errnum* come from *errno*, but *strerror*() shall map any value of type **int** to a message.

The application shall not modify the string returned. The returned string pointer might be invalidated or the string content might be overwritten by a subsequent call to *strerror*(), or by a subsequent call to *strerror_l*() in the same thread. The returned pointer and the string content might also be invalidated if the calling thread is terminated.

The string may be overwritten by a subsequent call to *strerror_l*() in the same thread.

The contents of the error message strings returned by *strerror*() should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *strerror*().

The *strerror*() and *strerror_l*() functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error of *strerror*(), an application wishing to check for error situations should set *errno* to 0, then call *strerror*(), then check *errno*. Similarly, since *strerror_l*() is required to return a string for some errors, an application wishing to check for all error situations should set *errno* to 0, then call *strerror_l*(), then check *errno*.

The *strerror*() function need not be thread-safe.

The *strerror_l*() function shall map the error number in *errnum* to a locale-dependent error message string in the locale represented by *locale* and shall return a pointer to it.

The *strerror_r*() function shall map the error number in *errnum* to a locale-dependent error message string and shall return the string in the buffer pointed to by *strerrbuf*, with length *buflen*.

If the value of *errnum* is a valid error number, the message string shall indicate what error occurred; if the value of *errnum* is zero, the message string shall either be an empty string or indicate that no error occurred; otherwise, if these functions complete successfully, the message string shall indicate that an unknown error occurred.

The behavior is undefined if the *locale* argument to *strerror_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, whether successful or not, *strerror*() shall return a pointer to the generated message string. On error *errno* may be set, but no return value is reserved to indicate an error.

Upon successful completion, *strerror_l*() shall return a pointer to the generated message string. If *errnum* is not a valid error number, *errno* may be set to **[EINVAL]**, but a pointer to a message string shall still be returned. If any other error occurs, *errno* shall be set to indicate the error and a null pointer shall be returned.

Upon successful completion, *strerror_r*() shall return 0. Otherwise, an error number shall be returned to indicate the error.

ERRORS

These functions may fail if:

EINVAL
The value of *errnum* is neither a valid error number nor zero.

The *strerror_r*() function may fail if:

ERANGE
Insufficient storage was supplied via *strerrbuf* and *buflen* to contain the generated message string.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Historically in some implementations, calls to *perror*() would overwrite the string that the pointer returned by *strerror*() points to. Such implementations did not conform to the ISO C standard; however, application developers should be aware of this behavior if they wish their applications to be portable to such implementations.

RATIONALE

The *strerror_l*() function is required to be thread-safe, thereby eliminating the need for an equivalent to the *strerror_r*() function.

Earlier versions of this standard did not explicitly require that the error message strings returned by *strerror*() and *strerror_r*() provide any information about the error. This version of the standard requires a meaningful message for any successful completion.

Since no return value is reserved to indicate a *strerror*() error, but all calls (whether successful or not) must return a pointer to a message string, on error *strerror*() can return a pointer to an empty string or a pointer to a meaningful string that can be printed.

Note that the **[EINVAL]** error condition is a may fail error. If an invalid error number is supplied as the value of *errnum*, applications should be prepared to handle any of the following:

1. Error (with no meaningful message): *errno* is set to **[EINVAL]**, the return value is a pointer to an empty string.
2. Successful completion: *errno* is unchanged and the return value points to a string like **"unknown-**error" or **"error**number**xxx"** (where *xxx* is the value of *errnum*).
3. Combination of #1 and #2: *errno* is set to **[EINVAL]** and the return value points to a string like **"unknown**error" or **"error**number**xxx"** (where *xxx* is the value of *errnum*). Since applications frequently use the return value of *strerror*() as an argument to functions like *fprintf*() (without checking the return value) and since applications have no way to parse an error message string to determine whether *errnum* represents a valid error number, implementations are encouraged to implement #3. Similarly, implementations are encouraged to have *strerror_r*() return **[EINVAL]** and put a string like **"unknown-**error" or **"error**number**xxx"** in the buffer pointed to by *strerrbuf* when the value of *errnum* is not a valid error number.

Some applications rely on being able to set *errno* to 0 before calling a function with no reserved value to indicate an error, then call *strerror*(*errno*) afterwards to detect whether an error occurred (because *errno* changed) or to indicate success (because *errno* remained zero). This usage pattern requires that *strerror*(0) succeed with useful results. Previous versions of the standard did not specify the behavior when *errnum* is zero.

FUTURE DIRECTIONS

None.

SEE ALSO

perror( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strfmon, strfmon_l — convert monetary value to a string

SYNOPSIS

```
#include <monetary.h>

ssize_t strfmon(char *restrict s, size_t maxsize,
   const char *restrict format, ...);
ssize_t strfmon_l(char *restrict s, size_t maxsize,
   locale_t locale, const char *restrict format, ...);
```

DESCRIPTION

The *strfmon*() function shall place characters into the array pointed to by *s* as controlled by the string pointed to by *format*. No more than *maxsize* bytes are placed into the array.

The format is a character string, beginning and ending in its initial state, if any, that contains two types of objects: *plain characters*, which are simply copied to the output stream, and *conversion specifications*, each of which shall result in the fetching of zero or more arguments which are converted and formatted. The results are undefined if there are insufficient arguments for the format. If the format is exhausted while arguments remain, the excess arguments are simply ignored.

The application shall ensure that a conversion specification consists of the following sequence:

* A **'%'** character
* Optional flags
* Optional field width
* Optional left precision
* Optional right precision
* A required conversion specifier character that determines the conversion to be performed

The *strfmon_l*() function shall be equivalent to the *strfmon*() function, except that the locale data used is from the locale represented by *locale*.

Flags

One or more of the following optional flags can be specified to control the conversion:

=*f* An **'='** followed by a single character *f* which is used as the numeric fill character. In order to work with precision or width counts, the fill character shall be a single byte character; if not, the behavior is undefined. The default numeric fill character is the <space>. This flag does not affect field width filling which always uses the <space>. This flag is ignored unless a left precision (see below) is specified.

^ Do not format the currency amount with grouping characters. The default is to insert the grouping characters if defined for the current locale.

+ or (Specify the style of representing positive and negative currency amounts. Only one of **'+'** or **'('** may be specified. If **'+'** is specified, the locale's equivalent of **'+'** and **'−'** are used (for example, in many locales, the empty string if positive and **'−'** if negative). If **'('** is specified, negative amounts are enclosed within parentheses. If neither flag is specified, the **'+'** style is used.

! Suppress the currency symbol from the output conversion.

− Specify the alignment. If this flag is present the result of the conversion is left-justified (padded to the right) rather than right-justified. This flag shall be ignored unless a field width (see below) is specified.

Field Width

w A decimal digit string *w* specifying a minimum field width in bytes in which the result of the conversion is right-justified (or left-justified if the flag **'−'** is specified). The default is 0.

Left Precision

#*n* A **'#'** followed by a decimal digit string *n* specifying a maximum number of digits expected to be formatted to the left of the radix character. This option can be used to keep the formatted output from multiple calls to the *strfmon*() function aligned in the same columns. It can also be used to fill unused positions with a special character as in **"$***123.45"**. This option causes an amount to be formatted as if it has the number of digits specified by *n*. If more than *n* digit positions are required, this conversion specification is ignored. Digit positions in excess of those actually required are filled with the numeric fill character (see the =*f* flag above).

If grouping has not been suppressed with the **'^'** flag, and it is defined for the current locale, grouping separators are inserted before the fill characters (if any) are added. Grouping separators are not applied to fill characters even if the fill character is a digit.

To ensure alignment, any characters appearing before or after the number in the formatted output such as currency or sign symbols are padded as necessary with <space> characters to make their positive and negative formats an equal length.

Right Precision

.*p* A <period> followed by a decimal digit string *p* specifying the number of digits after the radix character. If the value of the right precision *p* is 0, no radix character appears. If a right precision is not included, a default specified by the current locale is used. The amount being formatted is rounded to the specified number of digits prior to formatting.

Conversion Specifier Characters

The conversion specifier characters and their meanings are:

i The **double** argument is formatted according to the locale's international currency format (for example, in the US: USD 1,234.56). If the argument is ±Inf or NaN, the result of the conversion is unspecified.

n The **double** argument is formatted according to the locale's national currency format (for example, in the US: $1,234.56). If the argument is ±Inf or NaN, the result of the conversion is unspecified.

% Convert to a **'%'**; no argument is converted. The entire conversion specification shall be **%%**.

Locale Information

The *LC_MONETARY* category of the current locale affects the behavior of this function including the monetary radix character (which may be different from the numeric radix character affected by the *LC_NUMERIC* category), the grouping separator, the currency symbols, and formats. The international currency symbol should be conformant with the ISO 4217: 2001 standard.

If the value of *maxsize* is greater than {SSIZE_MAX}, the result is implementation-defined.

The behavior is undefined if the *locale* argument to *strfmon_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise, −1 shall be returned, the contents of the array are unspecified, and *errno* shall be set to indicate the error.

ERRORS

These functions shall fail if:

E2BIG Conversion stopped due to lack of space in the buffer.

The following sections are informative.

EXAMPLES

Given a locale for the US and the values 123.45, −123.45, and 3456.781, the following output might be produced. Square brackets ("[ ]") are used in this example to delimit the output.

```
%n          [$123.45]         Default formatting
            [-$123.45]
            [$3,456.78]

%11n        [    $123.45]     Right align within an 11-character field
            [   -$123.45]
            [  $3,456.78]

%#5n        [ $   123.45]     Aligned columns for values up to 99 999
            [-$   123.45]
            [ $ 3,456.78]

%=*#5n      [ $***123.45]     Specify a fill character
            [-$***123.45]
            [ $*3,456.78]

%=0#5n      [ $000123.45]     Fill characters do not use grouping
            [-$000123.45]     even if the fill character is a digit
            [ $03,456.78]

%^#5n       [ $  123.45]      Disable the grouping separator
            [-$  123.45]
            [ $ 3456.78]

%^#5.0n     [ $  123]         Round off to whole units
            [-$  123]
            [ $ 3457]

%^#5.4n     [ $  123.4500]    Increase the precision
            [-$  123.4500]
            [ $ 3456.7810]

%(#5n       [ $   123.45 ]    Use an alternative pos/neg style
            [($   123.45)]
            [ $ 3,456.78 ]

%!(#5n      [    123.45 ]     Disable the currency symbol
            [(   123.45)]
            [  3,456.78 ]

%-14#5.4n   [ $   123.4500 ]  Left-justify the output
            [-$   123.4500 ]
            [ $ 3,456.7810 ]

%14#5.4n    [  $   123.4500]  Corresponding right-justified output
            [ -$   123.4500]
            [  $ 3,456.7810]
```

See also the EXAMPLES section in *fprintf*().

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

Lowercase conversion characters are reserved for future standards use and uppercase for implementation-defined use.

SEE ALSO

fprintf( ), *localeconv*( )

The Base Definitions volume of POSIX.1-2017, **<monetary.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strftime, strftime_l — convert date and time to a string

SYNOPSIS

```
#include <time.h>

size_t strftime(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr);
size_t strftime_l(char *restrict s, size_t maxsize,
    const char *restrict format, const struct tm *restrict timeptr,
    locale_t locale);
```

DESCRIPTION

For *strftime*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strftime*() function shall place bytes into the array pointed to by *s* as controlled by the string pointed to by *format*. The format is a character string, beginning and ending in its initial shift state, if any. The format string consists of zero or more conversion specifications and ordinary characters.

Each conversion specification is introduced by the **'%'** character after which the following appear in sequence:

* An optional flag:

  0 The zero character (**'0'**), which specifies that the character used as the padding character is **'0'**,

  + The <plus-sign> character (**'+'**), which specifies that the character used as the padding character is **'0'**, and that if and only if the field being produced consumes more than four bytes to represent a year (for **%F**, **%G**, or **%Y**) or more than two bytes to represent the year divided by 100 (for **%C**) then a leading <plus-sign> character shall be included if the year being processed is greater than or equal to zero or a leading <hyphen-minus> character (**'−'**) shall be included if the year is less than zero.

  The default padding character is unspecified.

* An optional minimum field width. If the converted value, including any leading **'+'** or **'−'** sign, has fewer bytes than the minimum field width and the padding character is not the NUL character, the output shall be padded on the left (after any leading **'+'** or **'−'** sign) with the padding character.
* An optional **E** or **O** modifier.
* A terminating conversion specifier character that indicates the type of conversion to be applied.

The results are unspecified if more than one flag character is specified, a flag character is specified without a minimum field width; a minimum field width is specified without a flag character; a modifier is specified with a flag or with a minimum field width; or if a minimum field width is specified for any conversion specifier other than **C**, **F**, **G**, or **Y**.

All ordinary characters (including the terminating NUL character) are copied unchanged into the array. If copying takes place between objects that overlap, the behavior is undefined. No more than maxsize bytes are placed into the array. Each conversion specifier is replaced by appropriate characters as described in the following list. The appropriate characters are determined using the *LC_TIME* category of the current locale and by the values of zero or more members of the broken-down time structure pointed to by *timeptr*, as specified in brackets in the description. If any of the specified values are outside the normal range, the characters stored are unspecified.

The *strftime_l*() function shall be equivalent to the *strftime*() function, except that the locale data used is from the locale represented by *locale*.

Local timezone information is used as though *strftime*() called *tzset*().

The following conversion specifiers shall be supported:

a	Replaced by the locale's abbreviated weekday name. [*tm_wday*]
A	Replaced by the locale's full weekday name. [*tm_wday*]
b	Replaced by the locale's abbreviated month name. [*tm_mon*]
B	Replaced by the locale's full month name. [*tm_mon*]
c	Replaced by the locale's appropriate date and time representation. (See the Base Definitions volume of POSIX.1-2017, <*time.h*>.)
C	Replaced by the year divided by 100 and truncated to an integer, as a decimal number. [*tm_year*] If a minimum field width is not specified, the number of characters placed into the array pointed to by *s* will be the number of digits in the year divided by 100 or two, whichever is greater. If a minimum field width is specified, the number of characters placed into the array pointed to by *s* will be the number of digits in the year divided by 100 or the minimum field width, whichever is greater.
d	Replaced by the day of the month as a decimal number [01,31]. [*tm_mday*]
D	Equivalent to **%m/%d/%y**. [*tm_mon*, *tm_mday*, *tm_year*]
e	Replaced by the day of the month as a decimal number [1,31]; a single digit is preceded by a space. [*tm_mday*]
F	Equivalent to %+4Y-%m-%d if no flag and no minimum field width are specified. [*tm_year*, *tm_mon*, *tm_mday*] If a minimum field width of *x* is specified, the year shall be output as if by the **Y** specifier (described below) with whatever flag was given and a minimum field width of *x*−6. If *x* is less than 6, the behavior shall be as if *x* equalled 6. If the minimum field width is specified to be 10, and the year is four digits long, then the output string produced will match the ISO 8601: 2004 standard subclause 4.1.2.2 complete representation, extended format date representation of a specific day. If a + flag is specified, a minimum field width of *x* is specified, and *x*−7 bytes are sufficient to hold the digits of the year (not including any needed sign character), then the output will match the ISO 8601: 2004 standard subclause 4.1.2.4 complete representation, expanded format date representation of a specific day.
g	Replaced by the last 2 digits of the week-based year (see below) as a decimal number [00,99]. [*tm_year*, *tm_wday*, *tm_yday*]
G	Replaced by the week-based year (see below) as a decimal number (for example, 1977). [*tm_year*, *tm_wday*, *tm_yday*] If a minimum field width is specified, the number of characters placed into the array pointed to by *s* will be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
h	Equivalent to **%b**. [*tm_mon*]
H	Replaced by the hour (24-hour clock) as a decimal number [00,23]. [*tm_hour*]
I	Replaced by the hour (12-hour clock) as a decimal number [01,12]. [*tm_hour*]
j	Replaced by the day of the year as a decimal number [001,366]. [*tm_yday*]
m	Replaced by the month as a decimal number [01,12]. [*tm_mon*]
M	Replaced by the minute as a decimal number [00,59]. [*tm_min*]

n	Replaced by a <newline>.
p	Replaced by the locale's equivalent of either a.m. or p.m. [*tm_hour*]
r	Replaced by the time in a.m. and p.m. notation; in the POSIX locale this shall be equivalent to **%I**:**%M**:**%S %p**. [*tm_hour*, *tm_min*, *tm_sec*]
R	Replaced by the time in 24-hour notation (**%H**:**%M**). [*tm_hour*, *tm_min*]
S	Replaced by the second as a decimal number [00,60]. [*tm_sec*]
t	Replaced by a <tab>.
T	Replaced by the time (**%H**:**%M**:**%S**). [*tm_hour*, *tm_min*, *tm_sec*]
u	Replaced by the weekday as a decimal number [1,7], with 1 representing Monday. [*tm_wday*]
U	Replaced by the week number of the year as a decimal number [00,53]. The first Sunday of January is the first day of week 1; days in the new year before this are in week 0. [*tm_year*, *tm_wday*, *tm_yday*]
V	Replaced by the week number of the year (Monday as the first day of the week) as a decimal number [01,53]. If the week containing 1 January has four or more days in the new year, then it is considered week 1. Otherwise, it is the last week of the previous year, and the next week is week 1. Both January 4th and the first Thursday of January are always in week 1. [*tm_year*, *tm_wday*, *tm_yday*]
w	Replaced by the weekday as a decimal number [0,6], with 0 representing Sunday. [*tm_wday*]
W	Replaced by the week number of the year as a decimal number [00,53]. The first Monday of January is the first day of week 1; days in the new year before this are in week 0. [*tm_year*, *tm_wday*, *tm_yday*]
x	Replaced by the locale's appropriate date representation. (See the Base Definitions volume of POSIX.1-2017, <*time.h*>.)
X	Replaced by the locale's appropriate time representation. (See the Base Definitions volume of POSIX.1-2017, <*time.h*>.)
y	Replaced by the last two digits of the year as a decimal number [00,99]. [*tm_year*]
Y	Replaced by the year as a decimal number (for example, 1997). [*tm_year*] If a minimum field width is specified, the number of characters placed into the array pointed to by *s* will be the number of digits and leading sign characters (if any) in the year, or the minimum field width, whichever is greater.
z	Replaced by the offset from UTC in the ISO 8601:2004 standard format (**+hhmm** or **−hhmm**), or by no characters if no timezone is determinable. For example, **"-0430"** means 4 hours 30 minutes behind UTC (west of Greenwich). If *tm_isdst* is zero, the standard time offset is used. If *tm_isdst* is greater than zero, the daylight savings time offset is used. If *tm_isdst* is negative, no characters are returned. [*tm_isdst*]
Z	Replaced by the timezone name or abbreviation, or by no bytes if no timezone information exists. [*tm_isdst*]
%	Replaced by **%**.

If a conversion specification does not correspond to any of the above, the behavior is undefined.

If a **struct tm** broken-down time structure is created by *localtime*() or *localtime_r*(), or modified by *mktime*(), and the value of *TZ* is subsequently modified, the results of the **%Z** and **%z** *strftime*() conversion specifiers are undefined, when *strftime*() is called with such a broken-down time structure.

If a **struct tm** broken-down time structure is created or modified by *gmtime*() or *gmtime_r*(), it is unspecified whether the result of the **%Z** and **%z** conversion specifiers shall refer to UTC or the current local timezone, when *strftime*() is called with such a broken-down time structure.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the **E** or **O** modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist for the current locale (see ERA in the Base Definitions volume of POSIX.1-2017, *Section 7.3.5*, *LC_TIME*), the behavior shall be as if the unmodified conversion specification were used.

%Ec	Replaced by the locale's alternative appropriate date and time representation.
%EC	Replaced by the name of the base year (period) in the locale's alternative representation.
%Ex	Replaced by the locale's alternative date representation.
%EX	Replaced by the locale's alternative time representation.
%Ey	Replaced by the offset from **%EC** (year only) in the locale's alternative representation.
%EY	Replaced by the full alternative year representation.
%Od	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading zeros if there is any alternative symbol for zero; otherwise, with leading <space> characters.
%Oe	Replaced by the day of the month, using the locale's alternative numeric symbols, filled as needed with leading <space> characters.
%OH	Replaced by the hour (24-hour clock) using the locale's alternative numeric symbols.
%OI	Replaced by the hour (12-hour clock) using the locale's alternative numeric symbols.
%Om	Replaced by the month using the locale's alternative numeric symbols.
%OM	Replaced by the minutes using the locale's alternative numeric symbols.
%OS	Replaced by the seconds using the locale's alternative numeric symbols.
%Ou	Replaced by the weekday as a number in the locale's alternative representation (Monday=1).
%OU	Replaced by the week number of the year (Sunday as the first day of the week, rules corresponding to **%U**) using the locale's alternative numeric symbols.
%OV	Replaced by the week number of the year (Monday as the first day of the week, rules corresponding to **%V**) using the locale's alternative numeric symbols.
%Ow	Replaced by the number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
%OW	Replaced by the week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
%Oy	Replaced by the year (offset from **%C**) using the locale's alternative numeric symbols.

%g, **%G**, and **%V** give values according to the ISO 8601: 2004 standard week-based year. In this system, weeks begin on a Monday and week 1 of the year is the week that includes January 4th, which is also the week that includes the first Thursday of the year, and is also the first week that contains at least four days in the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year; thus, for Saturday 2nd January 1999, **%G** is replaced by 1998 and **%V** is replaced by 53. If December 29th, 30th, or 31st is a Monday, it and any following days are part of week 1 of the following year. Thus, for Tuesday 30th December 1997, **%G** is replaced by 1998 and **%V** is replaced by 01.

If a conversion specifier is not one of the above, the behavior is undefined.

The behavior is undefined if the *locale* argument to *strftime_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

If the total number of resulting bytes including the terminating null byte is not more than *maxsize*, these functions shall return the number of bytes placed into the array pointed to by *s*, not including the terminating NUL character. Otherwise, 0 shall be returned and the contents of the array are unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Getting a Localized Date String

The following example first sets the locale to the user's default. The locale information will be used in the *nl_langinfo*() and *strftime*() functions. The *nl_langinfo*() function returns the localized date string which specifies how the date is laid out. The *strftime*() function takes this information and, using the **tm** structure for values, places the date and time information into *datestring*.

```
#include <time.h>
#include <locale.h>
#include <langinfo.h>
...
struct tm *tm;
char datestring[256];
...
setlocale (LC_ALL, "");
...
strftime (datestring, sizeof(datestring), nl_langinfo (D_T_FMT), tm);
...
```

APPLICATION USAGE

The range of values for **%S** is [00,60] rather than [00,59] to allow for the occasional leap second.

Some of the conversion specifications are duplicates of others. They are included for compatibility with *nl_cxtime*() and *nl_ascxtime*(), which were published in Issue 2.

The **%C**, **%F**, **%G**, and **%Y** format specifiers in *strftime*() always print full values, but the *strptime*() **%C**, **%F**, and **%Y** format specifiers only scan two digits (assumed to be the first two digits of a four-digit year) for **%C** and four digits (assumed to be the entire (four-digit) year) for **%F** and **%Y**. This mimics the behavior of *printf*() and *scanf*(); that is:

```
printf("%2d", x = 1000);
```

prints **"1000"**, but:

```
scanf(%2d", &x);
```

when given **"1000"** as input will only store 10 in *x*). Applications using extended ranges of years must be sure that the number of digits specified for scanning years with *strptime*() matches the number of digits that will actually be present in the input stream. Historic implementations of the **%Y** conversion specification (with no flags and no minimum field width) produced different output formats. Some always produced at least four digits (with 0 fill for years from 0 through 999) while others only produced the number of digits present in the year (with no fill and no padding). These two forms can be produced with the **'0'** flag and a minimum field width options using the conversions specifications **%04Y** and **%01Y**, respectively.

In the past, the C and POSIX standards specified that **%F** produced an ISO 8601: 2004 standard date format, but didn't specify which one. For years in the range [0001,9999], POSIX.1-2008 requires that the output produced match the ISO 8601: 2004 standard complete representation extended format (YYYY-MM-DD) and for years outside of this range produce output that matches the ISO 8601: 2004 standard expanded representation extended format (<+/-><Underline>Y</Underline>YYYY-MM-DD). To fully meet ISO 8601: 2004 standard requirements, the producer and consumer must agree on a date format that has a specific number of bytes reserved to hold the characters used to represent the years that is sufficiently large to hold all values that will be shared. For example, the **%+13F** conversion specification will produce output matching the format **"<+/->YYYYYY-MM-DD"** (a leading **'+'** or **'−'** sign; a six-digit, 0-filled year; a **'−'**; a two-digit, leading 0-filled month; another **'−'**; and the two-digit, leading 0-filled day within the month).

Note that if the year being printed is greater than 9999, the resulting string from the unadorned **%F** conversion specifications will not conform to the ISO 8601: 2004 standard extended format, complete representation for a date and will instead be an extended format, expanded representation (presumably without the required agreement between the date's producer and consumer).

In the C or POSIX locale, the **E** and **O** modifiers are ignored and the replacement strings for the following specifiers are:

%a	The first three characters of **%A**.
%A	One of Sunday, Monday, . . ., Saturday.
%b	The first three characters of **%B**.
%B	One of January, February, . . ., December.
%c	Equivalent to **%a %b %e %T %Y**.
%p	One of AM or PM.
%r	Equivalent to **%I**:**%M**:**%S %p**.
%x	Equivalent to **%m/%d/%y**.
%X	Equivalent to **%T**.
%Z	Implementation-defined.

RATIONALE

The **%Y** conversion specification to *strftime*() was frequently assumed to be a four-digit year, but the ISO C standard does not specify that **%Y** is restricted to any subset of allowed values from the *tm_year* field. Similarly, the **%C** conversion specification was assumed to be a two-digit field and the first part of the output from the **%F** conversion specification was assumed to be a four-digit field. With *tm_year* being a signed 32 or more-bit **int** and with many current implementations supporting 64-bit **time_t** types in one or more programming environments, these assumptions are clearly wrong.

POSIX.1-2008 now allows the format specifications **%0xC**, **%0xF**, **%0xG**, and **%0xY** (where **'x'** is a string of decimal digits used to specify printing and scanning of a string of *x* decimal digits) with leading zero fill characters. Allowing applications to set the field width enables them to agree on the number of digits to be printed and scanned in the ISO 8601: 2004 standard expanded representation of a year (for **%F**, **%G**, and **%Y**) or all but the last two digits of the year (for **%C**). This is based on a feature in some versions of GNU **libc**'s *strftime*(). The GNU version allows specifying space, zero, or no-fill characters in *strftime*() format strings, but does not allow any flags to be specified in *strptime*() format strings. These implementations also allow these flags to be specified for any numeric field. POSIX.1-2008 only requires the zero fill flag (**'0'**) and only requires that it be recognized when processing **%C**, **%F**, **%G**, and **%Y** specifications when a minimum field width is also specified. The **'0'** flag is the only flag needed to produce and scan the ISO 8601: 2004 standard year fields using the extended format forms. POSIX.1-2008 also allows applications to specify the same flag and field width specifiers to be used in both *strftime*() and *strptime*() format strings for symmetry. Systems may provide other flag characters and may accept flags in conjunction with conversion specifiers other than **%C**, **%F**, **%G**, and **%Y**; but portable applications cannot depend on such extensions.

POSIX.1-2008 now also allows the format specifications **%+xC**, **%+xF**, **%+xG**, and **%+xY** (where **'x'** is a string of decimal digits used to specify printing and scanning of a string of **'x'** decimal digits) with leading zero fill characters and a leading **'+'** sign character if the year being converted is more than four digits or a minimum field width is specified that allows room for more than four digits for the year. This allows date providers and consumers to agree on a specific number of digits to represent a year as required by the ISO 8601: 2004 standard expanded representation formats. The expanded representation formats all require the year to begin with a leading **'+'** or **'−'** sign. (All of these specifiers can also provide a leading **'−'** sign for negative years. Since negative years and the year 0 don't fit well with the Gregorian or Julian calendars, the normal ranges of dates start with year 1. The ISO C standard allows *tm_year* to assume values corresponding to years before year 1, but the use of such years provided unspecified results.)

Some earlier version of this standard specified that applications wanting to use *strptime*() to scan dates and times printed by *strftime*() should provide non-digit characters between fields to separate years from months and days. It also supported **%F** to print and scan the ISO 8601: 2004 standard extended format, complete representation date for years 1 through 9999 (i.e., YYYY-MM-DD). However, many applications were written to print (using *strftime*()) and scan (using *strptime*()) dates written using the basic format complete representation (four-digit years) and truncated representation (two-digit years) specified by the ISO 8601: 2004 standard representation of dates and times which do not have any separation characters between fields. The ISO 8601: 2004 standard also specifies basic format expanded representation where the creator and consumer of these fields agree beforehand to represent years as leading zero-filled strings of an agreed length of more than four digits to represent a year (again with no separation characters when year, month, and day are all displayed). Applications producing and consuming expanded representations are encouraged to use the **'+'** flag and an appropriate maximum field width to scan the year including the leading sign. Note that even without the **'+'** flag, years less than zero may be represented with a leading <hyphen-minus> for **%F**, **%G**, and **%Y** conversion specifications. Using negative years results in unspecified behavior.

If a format specification **%+xF** with the field width *x* greater than 11 is specified and the width is large enough to display the full year, the output string produced will match the ISO 8601: 2004 standard subclause 4.1.2.4 expanded representation, extended format date representation for a specific day. (For years in the range [1,99 999], **%+12F** is sufficient for an agreed five-digit year with a leading sign using the ISO 8601: 2004 standard expanded representation, extended format for a specific day **"<+/->YYYYY-MM-DD"**.) Note also that years less than 0 may produce a leading <hyphen-minus> character (**'−'**) when using **%Y** or **%C** whether or not the **'0'** or **'+'** flags are used.

The difference between the **'0'** flag and the **'+'** flag is whether the leading **'+'** character will be provided for years >9999 as required for the ISO 8601: 2004 standard extended representation format containing a year. For example:

box center tab(!); cB | cB | cB | cB cB | cB | cB | cB l | lf5 | l | l. !!*strftime*()!*strptime*() Year!Conversion Specification!Output!Scan Back _ 1970!%Y!1970!1970 _ 1970!%+4Y!1970!1970 _ 27!%Y!27 or 0027!27 _ 270!%Y!270 or 0270!270 _ 270!%+4Y!0270!270 _ 17!%C%y!0017!17 _ 270!%C%y!0270!270 _ 12345!%Y!12345!1234* _ 12345!%+4Y!+12345!123* _ 12345!%05Y!12345!12345 _ 270!%+5Y or %+3C%y!+0270!270 _ 12345!%+5Y or %+3C%y!+12345!1234* _ 12345!%06Y or %04C%y!012345!12345 _ 12345!%+6Y or %+4C%y!+12345!12345 _ 123456!%08Y or %06C%y!00123456!123456 _ 123456!%+8Y or %+6C%y!+0123456!123456

In the cases above marked with a * in the *strptime*() scan back field, the implied or specified number of characters scanned by *strptime*() was less than the number of characters output by *strftime*() using the same format; so the remaining digits of the year were dropped when the output date produced by *strftime*() was scanned back in by *strptime*().

FUTURE DIRECTIONS

None.

SEE ALSO

asctime( ), *clock*( ), *ctime*( ), *difftime*( ), *getdate*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strptime*( ), *time*( ), *tzset*( ), *uselocale*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, *Section 7.3.5*, *LC_TIME*, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strlen, strnlen — get length of fixed size string

SYNOPSIS

```
#include <string.h>

size_t strlen(const char *s);
size_t strnlen(const char *s, size_t maxlen);
```

DESCRIPTION

For *strlen*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strlen*() function shall compute the number of bytes in the string to which *s* points, not including the terminating NUL character.

The *strnlen*() function shall compute the smaller of the number of bytes in the array to which *s* points, not including any terminating NUL character, or the value of the *maxlen* argument. The *strnlen*() function shall never examine more than *maxlen* bytes of the array pointed to by *s*.

RETURN VALUE

The *strlen*() function shall return the length of *s*; no return value shall be reserved to indicate an error.

The *strnlen*() function shall return the number of bytes preceding the first null byte in the array to which *s* points, if *s* contains a null byte within the first *maxlen* bytes; otherwise, it shall return *maxlen*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Getting String Lengths

The following example sets the maximum length of *key* and *data* by using *strlen*() to get the lengths of those strings.

```
#include <string.h>
...
struct element {
    char *key;
    char *data;
};
...
char *key, *data;
int len;

*keylength = *datalength = 0;
...
if ((len = strlen(key)) > *keylength)
    *keylength = len;
if ((len = strlen(data)) > *datalength)
    *datalength = len;
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcslen( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strncasecmp, strncasecmp_l — case-insensitive string comparisons

SYNOPSIS

```
#include <strings.h>

int strncasecmp(const char *s1, const char *s2, size_t n);
int strncasecmp_l(const char *s1, const char *s2,
    size_t n, locale_t locale);
```

DESCRIPTION

Refer to *strcasecmp*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strncat — concatenate a string with part of another

SYNOPSIS

#include <string.h>

char *strncat(char *restrict *s1*, const char *restrict *s2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strncat*() function shall append not more than *n* bytes (a NUL character and bytes that follow it are not appended) from the array pointed to by *s2* to the end of the string pointed to by *s1*. The initial byte of *s2* overwrites the NUL character at the end of *s1*. A terminating NUL character is always appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *strncat*() function shall return *s1*; no return value shall be reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcat( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strncmp — compare part of two strings

SYNOPSIS

#include <string.h>

int strncmp(const char **s1*, const char **s2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strncmp*() function shall compare not more than *n* bytes (bytes that follow a NUL character are not compared) from the array pointed to by *s1* to the array pointed to by *s2*.

The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes (both interpreted as type **unsigned char**) that differ in the strings being compared.

RETURN VALUE

Upon successful completion, *strncmp*() shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *s1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *s2* respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

stpncpy, strncpy — copy fixed length string, returning a pointer to the array end

SYNOPSIS

```
#include <string.h>

char *stpncpy(char *restrict s1, const char *restrict s2, size_t n);
char *strncpy(char *restrict s1, const char *restrict s2, size_t n);
```

DESCRIPTION

For *strncpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *stpncpy*() and *strncpy*() functions shall copy not more than *n* bytes (bytes that follow a NUL character are not copied) from the array pointed to by *s2* to the array pointed to by *s1*.

If the array pointed to by *s2* is a string that is shorter than *n* bytes, NUL characters shall be appended to the copy in the array pointed to by *s1*, until *n* bytes in all are written.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

If a NUL character is written to the destination, the *stpncpy*() function shall return the address of the first such NUL character. Otherwise, it shall return *&s1*[*n*].

The *strncpy*() function shall return *s1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications must provide the space in *s1* for the *n* bytes to be transferred, as well as ensure that the *s2* and *s1* arrays do not overlap.

Character movement is performed differently in different implementations. Thus, overlapping moves may yield surprises.

If there is no NUL character byte in the first *n* bytes of the array pointed to by *s2*, the result is not null-terminated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcpy( ), *wcsncpy*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strndup — duplicate a specific number of bytes from a string

SYNOPSIS

```
#include <string.h>

char *strndup(const char *s, size_t size);
```

DESCRIPTION

Refer to *strdup*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strnlen — get length of fixed size string

SYNOPSIS

```
#include <string.h>

size_t strnlen(const char *s, size_t maxlen);
```

DESCRIPTION

Refer to *strlen*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strpbrk — scan a string for a byte

SYNOPSIS

#include <string.h>

char *strpbrk(const char **s1*, const char **s2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strpbrk*() function shall locate the first occurrence in the string pointed to by *s1* of any byte from the string pointed to by *s2*.

RETURN VALUE

Upon successful completion, *strpbrk*() shall return a pointer to the byte or a null pointer if no byte from *s2* occurs in *s1*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( ), *strrchr*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strptime — date and time conversion

SYNOPSIS

```
#include <time.h>

char *strptime(const char *restrict buf, const char *restrict format,
    struct tm *restrict tm);
```

DESCRIPTION

The *strptime*() function shall convert the character string pointed to by *buf* to values which are stored in the **tm** structure pointed to by *tm*, using the format specified by *format*.

The format is composed of zero or more directives. Each directive is composed of one of the following: one or more white-space characters (as specified by *isspace*()); an ordinary character (neither **'%'** nor a white-space character); or a conversion specification.

Each conversion specification is introduced by the **'%'** character after which the following appear in sequence:

* An optional flag, the zero character (**'0'**) or the <plus-sign> character (**'+'**), which is ignored.
* An optional field width. If a field width is specified, it shall be interpreted as a string of decimal digits that will determine the maximum number of bytes converted for the conversion rather than the number of bytes specified below in the description of the conversion specifiers.
* An optional **E** or **O** modifier.
* A terminating conversion specifier character that indicates the type of conversion to be applied.

The conversions are determined using the *LC_TIME* category of the current locale. The application shall ensure that there is white-space or other non-alphanumeric characters between any two conversion specifications unless all of the adjacent conversion specifications convert a known, fixed number of characters. In the following list, the maximum number of characters scanned (excluding the one matching the next directive) is as follows:

* If a maximum field width is specified, then that number
* Otherwise, the pattern **"{x}"** indicates that the maximum is *x*
* Otherwise, the pattern **"[x,y]"** indicates that the value shall fall within the range given (both bounds being inclusive), and the maximum number of characters scanned shall be the maximum required to represent any value in the range without leading zeros and without a leading <plus-sign>

The following conversion specifiers are supported.

The results are unspecified if a modifier is specified with a flag or with a minimum field width, or if a field width is specified for any conversion specifier other than **C** or **Y**.

a	The day of the week, using the locale's weekday names; either the abbreviated or full name may be specified.
A	Equivalent to **%a**.
b	The month, using the locale's month names; either the abbreviated or full name may be specified.
B	Equivalent to **%b**.
c	Replaced by the locale's appropriate date and time representation.
C	All but the last two digits of the year {2}; leading zeros shall be permitted but shall not be required. A leading **'+'** or **'−'** character shall be permitted before any leading zeros but shall not be required.

d	The day of the month [01,31]; leading zeros shall be permitted but shall not be required.
D	The date as **%m/%d/%y**.
e	Equivalent to **%d**.
h	Equivalent to **%b**.
H	The hour (24-hour clock) [00,23]; leading zeros shall be permitted but shall not be required.
I	The hour (12-hour clock) [01,12]; leading zeros shall be permitted but shall not be required.
j	The day number of the year [001,366]; leading zeros shall be permitted but shall not be required.
m	The month number [01,12]; leading zeros shall be permitted but shall not be required.
M	The minute [00,59]; leading zeros shall be permitted but shall not be required.
n	Any white space.
p	The locale's equivalent of a.m. or p.m.
r	12-hour clock time using the AM/PM notation if **t_fmt_ampm** is not an empty string in the *LC_TIME* portion of the current locale; in the POSIX locale, this shall be equivalent to **%I**:**%M**:**%S %p**.
R	The time as **%H**:**%M**.
S	The seconds [00,60]; leading zeros shall be permitted but shall not be required.
t	Any white space.
T	The time as **%H**:**%M**:**%S**.
U	The week number of the year (Sunday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
w	The weekday as a decimal number [0,6], with 0 representing Sunday.
W	The week number of the year (Monday as the first day of the week) as a decimal number [00,53]; leading zeros shall be permitted but shall not be required.
x	The date, using the locale's date format.
X	The time, using the locale's time format.
y	The last two digits of the year. When *format* contains neither a **C** conversion specifier nor a **Y** conversion specifier, values in the range [69,99] shall refer to years 1969 to 1999 inclusive and values in the range [00,68] shall refer to years 2000 to 2068 inclusive; leading zeros shall be permitted but shall not be required. A leading **'+'** or **'−'** character shall be permitted before any leading zeros but shall not be required. **Note:** It is expected that in a future version of this standard the default century inferred from a 2-digit year will change. (This would apply to all commands accepting a 2-digit year as input.)
Y	The full year {4}; leading zeros shall be permitted but shall not be required. A leading **'+'** or **'−'** character shall be permitted before any leading zeros but shall not be required.
%	Replaced by **%**.

Modified Conversion Specifiers

Some conversion specifiers can be modified by the **E** and **O** modifier characters to indicate that an alternative format or specification should be used rather than the one normally used by the unmodified conversion specifier. If the alternative format or specification does not exist in the current locale, the behavior shall be as if the unmodified conversion specification were used.

%Ec	The locale's alternative appropriate date and time representation.
%EC	The name of the base year (period) in the locale's alternative representation.

%Ex	The locale's alternative date representation.
%EX	The locale's alternative time representation.
%Ey	The offset from **%EC** (year only) in the locale's alternative representation.
%EY	The full alternative year representation.
%Od	The day of the month using the locale's alternative numeric symbols; leading zeros shall be permitted but shall not be required.
%Oe	Equivalent to **%Od**.
%OH	The hour (24-hour clock) using the locale's alternative numeric symbols.
%OI	The hour (12-hour clock) using the locale's alternative numeric symbols.
%Om	The month using the locale's alternative numeric symbols.
%OM	The minutes using the locale's alternative numeric symbols.
%OS	The seconds using the locale's alternative numeric symbols.
%OU	The week number of the year (Sunday as the first day of the week) using the locale's alternative numeric symbols.
%Ow	The number of the weekday (Sunday=0) using the locale's alternative numeric symbols.
%OW	The week number of the year (Monday as the first day of the week) using the locale's alternative numeric symbols.
%Oy	The year (offset from **%C**) using the locale's alternative numeric symbols.

A conversion specification composed of white-space characters is executed by scanning input up to the first character that is not white-space (which remains unscanned), or until no more characters can be scanned.

A conversion specification that is an ordinary character is executed by scanning the next character from the buffer. If the character scanned from the buffer differs from the one comprising the directive, the directive fails, and the differing and subsequent characters remain unscanned.

A series of conversion specifications composed of **%n**, **%t**, white-space characters, or any combination is executed by scanning up to the first character that is not white space (which remains unscanned), or until no more characters can be scanned.

Any other conversion specification is executed by scanning characters until a character matching the next directive is scanned, or until no more characters can be scanned. These characters, except the one matching the next directive, are then compared to the locale values associated with the conversion specifier. If a match is found, values for the appropriate **tm** structure members are set to values corresponding to the locale information. Case is ignored when matching items in *buf* such as month or weekday names. If no match is found, *strptime*() fails and no more characters are scanned.

RETURN VALUE

Upon successful completion, *strptime*() shall return a pointer to the character following the last character parsed. Otherwise, a null pointer shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Convert a Date-Plus-Time String to Broken-Down Time and Then into Seconds

The following example demonstrates the use of *strptime*() to convert a string into broken-down time. The broken-down time is then converted into seconds since the Epoch using *mktime*().

```
#include <time.h>
...
```

```
struct tm tm;
time_t t;

if (strptime("6 Dec 2001 12:33:45", "%d %b %Y %H:%M:%S", &tm) == NULL)
   /* Handle error */;

printf("year: %d; month: %d; day: %d;\n",
       tm.tm_year, tm.tm_mon, tm.tm_mday);
printf("hour: %d; minute: %d; second: %d\n",
       tm.tm_hour, tm.tm_min, tm.tm_sec);
printf("week day: %d; year day: %d\n", tm.tm_wday, tm.tm_yday);

tm.tm_isdst = -1;      /* Not set by strptime(); tells mktime()
                          to determine whether daylight saving time
                          is in effect */
t = mktime(&tm);
if (t == -1)
   /* Handle error */;
printf("seconds since the Epoch: %ld\n", (long) t);"
```

APPLICATION USAGE

Several "equivalent to" formats and the special processing of white-space characters are provided in order to ease the use of identical *format* strings for *strftime*() and *strptime*().

It should be noted that dates constructed by the *strftime*() function with the **%Y** or **%C%y** conversion specifiers may have values larger than 9 999. If the *strptime*() function is used to read such values using **%C%y** or **%Y**, the year values will be truncated to four digits. Applications should use **%+*w*%y** or **%+*x*Y** with *w* and *x* set large enough to contain the full value of any years that will be printed or scanned.

See also the APPLICATION USAGE section in *strftime*( ).

It is unspecified whether multiple calls to *strptime*() using the same **tm** structure will update the current contents of the structure or overwrite all contents of the structure. Conforming applications should make a single call to *strptime*() with a format and all data needed to completely specify the date and time being converted.

RATIONALE

See the RATIONALE section for *strftime*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

fprintf( ), *fscanf*( ), *strftime*( ), *time*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strrchr — string scanning operation

SYNOPSIS

#include <string.h>

char *strrchr(const char **s*, int *c*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strrchr*() function shall locate the last occurrence of *c* (converted to a **char**) in the string pointed to by *s*. The terminating NUL character is considered to be part of the string.

RETURN VALUE

Upon successful completion, *strrchr*() shall return a pointer to the byte or a null pointer if *c* does not occur in the string.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Finding the Base Name of a File

The following example uses *strrchr*() to get a pointer to the base name of a file. The *strrchr*() function searches backwards through the name of the file to find the last **'/'** character in *name*. This pointer (plus one) will point to the base name of the file.

```
#include <string.h>
...
const char *name;
char *basename;
...
basename = strrchr(name, '/') + 1;
...
```

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strsignal — get name of signal

SYNOPSIS

#include <string.h>

char *strsignal(int *signum*);

DESCRIPTION

The *strsignal*() function shall map the signal number in *signum* to an implementation-defined string and shall return a pointer to it. It shall use the same set of messages as the *psignal*() function.

The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *strsignal*() or *setlocale*(). The returned pointer might also be invalidated if the calling thread is terminated.

The contents of the message strings returned by *strsignal*() should be determined by the setting of the *LC_MESSAGES* category in the current locale.

The implementation shall behave as if no function defined in this standard calls *strsignal*().

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strsignal*(), then check *errno*.

The *strsignal*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, *strsignal*() shall return a pointer to a string. Otherwise, if *signum* is not a valid signal number, the return value is unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If *signum* is not a valid signal number, some implementations return NULL, while for others the *strsignal*() function returns a pointer to a string containing an unspecified message denoting an unknown signal. POSIX.1-2008 leaves this return value unspecified.

FUTURE DIRECTIONS

None.

SEE ALSO

psiginfo( ), *setlocale*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strspn — get length of a substring

SYNOPSIS

```
#include <string.h>

size_t strspn(const char *s1, const char *s2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strspn*() function shall compute the length (in bytes) of the maximum initial segment of the string pointed to by *s1* which consists entirely of bytes from the string pointed to by *s2*.

RETURN VALUE

The *strspn*() function shall return the computed length; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcspn( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strstr — find a substring

SYNOPSIS

#include <string.h>

char *strstr(const char **s1*, const char **s2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strstr*() function shall locate the first occurrence in the string pointed to by *s1* of the sequence of bytes (excluding the terminating NUL character) in the string pointed to by *s2*.

RETURN VALUE

Upon successful completion, *strstr*() shall return a pointer to the located string or a null pointer if the string is not found.

If *s2* points to a string with zero length, the function shall return *s1*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strchr( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtod, strtof, strtold — convert a string to a double-precision number

SYNOPSIS

#include <stdlib.h>

double strtod(const char *restrict *nptr*, char **restrict *endptr*);
float strtof(const char *restrict *nptr*, char **restrict *endptr*);
long double strtold(const char *restrict *nptr*, char **restrict *endptr*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*())
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional **'+'** or **'−'** sign, then one of the following:

* A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part consisting of the character **'e'** or the character **'E'**, optionally followed by a **'+'** or **'−'** character, and then followed by one or more decimal digits

* A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part consisting of the character **'p'** or the character **'P'**, optionally followed by a **'+'** or **'−'** character, and then followed by one or more decimal digits

* One of INF or INFINITY, ignoring case

* One of NAN or NAN(*n-char-sequence*$_{opt}$), ignoring case in the NAN part, where:

```
n-char-sequence:
   digit
   nondigit
   n-char-sequence digit
   n-char-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character, that is of the expected form. The subject sequence contains no characters if the input string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of characters starting with the first digit or the decimal-point character (whichever occurs first) shall be interpreted as a floating constant of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero is assumed to follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the sequence shall be interpreted as negated. A character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A character sequence NAN or NAN(*n-char-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-char sequences is implementation-defined. A pointer to the final string is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the value resulting from the conversion is correctly rounded.

The radix character is defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> (**'.'**).

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* is stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *strtod*(), *strtof*(), or *strtold*(), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value. If no conversion could be performed, 0 shall be returned, and *errno* may be set to **[EINVAL]**.

If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to **[ERANGE]**.

If the correct value would cause an underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type shall be returned and *errno* set to **[ERANGE]**.

ERRORS

These functions shall fail if:

ERANGE
The value to be returned would cause overflow or underflow.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in <*float.h*>) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy $L <= D <= U$. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

The changes to *strtod*() introduced by the ISO/IEC 9899: 1999 standard can alter the behavior of well-formed applications complying with the ISO/IEC 9899: 1990 standard and thus earlier versions of this standard. One such example would be:

```
int
what_kind_of_number (char *s)
{
    char *endp;
    double d;
    long l;

    d = strtod(s, &endp);
    if (s != endp && *endp == `\0')
        printf("It's a float with value %g\n", d);
    else
    {
        l = strtol(s, &endp, 0);
        if (s != endp && *endp == `\0')
            printf("It's an integer with value %ld\n", 1);
        else
            return 1;
    }
    return 0;
}
```

If the function is called with:

```
what_kind_of_number ("0x10")
```

an ISO/IEC 9899: 1990 standard-compliant library will result in the function printing:

```
It's an integer with value 16
```

With the ISO/IEC 9899: 1999 standard, the result is:

```
It's a float with value 16
```

The change in behavior is due to the inclusion of floating-point numbers in hexadecimal notation without requiring that either a decimal point or the binary exponent be present.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *isspace*( ), *localeconv*( ), *setlocale*( ), *strtol*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<float.h>**, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtoimax, strtoumax — convert string to integer type

SYNOPSIS

```
#include <inttypes.h>

intmax_t strtoimax(const char *restrict nptr, char **restrict endptr,
    int base);
uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
    int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall be equivalent to the *strtol*(), *strtoll*(), *strtoul*(), and *strtoull*() functions, except that the initial portion of the string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned and *errno* may be set to **[EINVAL]**.

If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to **[EINVAL]**.

If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for an **[EINVAL]** error before examining **endptr*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strtol( ), *strtoul*( )

The Base Definitions volume of POSIX.1-2017, **<inttypes.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer’s Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtok, strtok_r — split string into tokens

SYNOPSIS

```
#include <string.h>

char *strtok(char *restrict s, const char *restrict sep);
char *strtok_r(char *restrict s, const char *restrict sep,
   char **restrict state);
```

DESCRIPTION

For *strtok*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

A sequence of calls to *strtok*() breaks the string pointed to by *s* into a sequence of tokens, each of which is delimited by a byte from the string pointed to by *sep*. The first call in the sequence has *s* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *sep* may be different from call to call.

The first call in the sequence searches the string pointed to by *s* for the first byte that is *not* contained in the current separator string pointed to by *sep*. If no such byte is found, then there are no tokens in the string pointed to by *s* and *strtok*() shall return a null pointer. If such a byte is found, it is the start of the first token.

The *strtok*() function then searches from there for a byte that *is* contained in the current separator string. If no such byte is found, the current token extends to the end of the string pointed to by *s*, and subsequent searches for a token shall return a null pointer. If such a byte is found, it is overwritten by a NUL character, which terminates the current token. The *strtok*() function saves a pointer to the following byte, from which the next search for a token shall start.

Each subsequent call, with a null pointer as the value of the first argument, starts searching from the saved pointer and behaves as described above.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *strtok*().

The *strtok*() function need not be thread-safe.

The *strtok_r*() function shall be equivalent to *strtok*(), except that *strtok_r*() shall be thread-safe and the argument *state* points to a user-provided pointer that allows *strtok_r*() to maintain state between calls which scan the same string. The application shall ensure that the pointer pointed to by *state* is unique for each string (*s*) being processed concurrently by *strtok_r*() calls. The application need not initialize the pointer pointed to by *state* to any particular value. The implementation shall not update the pointer pointed to by *state* to point (directly or indirectly) to resources, other than within the string *s*, that need to be freed or released by the caller.

RETURN VALUE

Upon successful completion, *strtok*() shall return a pointer to the first byte of a token. Otherwise, if there is no token, *strtok*() shall return a null pointer.

The *strtok_r*() function shall return a pointer to the token found, or a null pointer when no token is found.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Searching for Word Separators

The following example searches for tokens separated by <space> characters.

```
#include <string.h>
...
char *token;
char line[] = "LINE TO BE SEPARATED";
char *search = " ";

/* Token will point to "LINE". */
token = strtok(line, search);

/* Token will point to "TO". */
token = strtok(NULL, search);
```

Find First two Fields in a Buffer

The following example uses *strtok*() to find two character strings (a key and data associated with that key) separated by any combination of <space>, <tab>, or <newline> characters at the start of the array of characters pointed to by *buffer*.

```
#include <string.h>
...
char    *buffer;
...
struct element {
    char *key;
    char *data;
} e;
...
// Load the buffer...
...
// Get the key and its data...
e.key = strtok(buffer, " \t\n");
e.data = strtok(NULL, " \t\n");
// Process the rest of the contents of the buffer...
...
```

APPLICATION USAGE

Note that if *sep* is the empty string, *strtok*() and *strtok_r*() return a pointer to the remainder of the string being tokenized.

The *strtok_r*() function is thread-safe and stores its state in a user-supplied buffer instead of possibly using a static data area that may be overwritten by an unrelated call from another thread.

RATIONALE

The *strtok*() function searches for a separator string within a larger string. It returns a pointer to the last substring between separator strings. This function uses static storage to keep track of the current string position between calls. The new function, *strtok_r*(), takes an additional argument, *state*, to keep track of the current position in the string.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtol, strtoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long strtol(const char *restrict nptr, char **restrict endptr, int base);
long long strtoll(const char *restrict nptr, char **restrict endptr,
   int base)
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *nptr* to a type **long** and **long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*())
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string.

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a **'+'** or **'−'** sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix **'0'** optionally followed by a sequence of the digits **'0'** to **'7'** only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters **'a'** (or **'A'**) to **'f'** (or **'F'**) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a **'+'** or **'−'** sign. The letters from **'a'** (or **'A'**) to **'z'** (or **'Z'**) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion is performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0, {LONG_MIN} or {LLONG_MIN}, and {LONG_MAX} or {LLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *strtol*() or *strtoll*(), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to **[EINVAL]**.

If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to **[EINVAL]**.

If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX}, {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and *errno* set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for an **[EINVAL]** error before examining **endptr*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *isalpha*( ), *strtod*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtold — convert a string to a double-precision number

SYNOPSIS

#include <stdlib.h>

long double strtold(const char *restrict *nptr*, char **restrict *endptr*);

DESCRIPTION

Refer to *strtod*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtoll — convert a string to a long integer

SYNOPSIS

```
#include <stdlib.h>

long long strtoll(const char *restrict str, char **restrict endptr,
    int base);
```

DESCRIPTION

Refer to *strtol*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtoul, strtoull — convert a string to an unsigned long

SYNOPSIS

```
#include <stdlib.h>

unsigned long strtoul(const char *restrict str,
   char **restrict endptr, int base);
unsigned long long strtoull(const char *restrict str,
   char **restrict endptr, int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall convert the initial portion of the string pointed to by *str* to a type **unsigned long** and **unsigned long long** representation, respectively. First, they decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space characters (as specified by *isspace*())
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final string of one or more unrecognized characters, including the terminating NUL character of the input string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If the value of *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a **'+'** or **'−'** sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix **'0'** optionally followed by a sequence of the digits **'0'** to **'7'** only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters **'a'** (or **'A'**) to **'f'** (or **'F'**) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a **'+'** or **'−'** sign. The letters from **'a'** (or **'A'**) to **'z'** (or **'Z'**) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* are permitted. If the value of *base* is 16, the characters 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input string, starting with the first non-white-space character that is of the expected form. The subject sequence shall contain no characters if the input string is empty or consists entirely of white-space characters, or if the first non-white-space character is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and the value of *base* is 0, the sequence of characters starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *str* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *strtoul*() or *strtoull*(), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to **[EINVAL]**.

If the value of *base* is not supported, 0 shall be returned and *errno* shall be set to **[EINVAL]**.

If the correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX} shall be returned and *errno* set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since the value of **endptr* is unspecified if the value of *base* is not supported, applications should either ensure that *base* has a supported value (0 or between 2 and 36) before the call, or check for an **[EINVAL]** error before examining **endptr*.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *isalpha*( ), *strtod*( ), *strtol*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strtoumax — convert a string to an integer type

SYNOPSIS

```
#include <inttypes.h>

uintmax_t strtoumax(const char *restrict nptr, char **restrict endptr,
   int base);
```

DESCRIPTION

Refer to *strtoimax*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

strxfrm, strxfrm_l — string transformation

SYNOPSIS

```
#include <string.h>

size_t strxfrm(char *restrict s1, const char *restrict s2, size_t n);
size_t strxfrm_l(char *restrict s1, const char *restrict s2,
   size_t n, locale_t locale);
```

DESCRIPTION

For *strxfrm*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *strxfrm*() and *strxfrm_l*() functions shall transform the string pointed to by *s2* and place the resulting string into the array pointed to by *s1*. The transformation is such that if *strcmp*() is applied to two transformed strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *strcoll*() or *strcoll_l*(), respectively, applied to the same two original strings with the same locale. No more than *n* bytes are placed into the resulting array pointed to by *s1*, including the terminating NUL character. If *n* is 0, *s1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

The *strxfrm*() and *strxfrm_l*() functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *strxfrm*() or *strxfrm_l*(), then check *errno*.

The behavior is undefined if the *locale* argument to *strxfrm_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *strxfrm*() and *strxfrm_l*() shall return the length of the transformed string (not including the terminating NUL character). If the value returned is *n* or more, the contents of the array pointed to by *s1* are unspecified.

On error, *strxfrm*() and *strxfrm_l*() may set *errno* but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

EINVAL
The string pointed to by the *s2* argument contains characters outside the domain of the collating sequence.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed strings can be ordered by *strcmp*() as appropriate to collating sequence information in the current locale (category *LC_COLLATE*).

The fact that when *n* is 0 *s1* is permitted to be a null pointer is useful to determine the size of the *s1* array prior to making the transformation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcmp( ), *strcoll*( )

The Base Definitions volume of POSIX.1-2017, **<string.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

swab — swap bytes

SYNOPSIS

```
#include <unistd.h>

void swab(const void *restrict src, void *restrict dest,
   ssize_t nbytes);
```

DESCRIPTION

The *swab*() function shall copy *nbytes* bytes, which are pointed to by *src*, to the object pointed to by *dest*, exchanging adjacent bytes. The *nbytes* argument should be even. If *nbytes* is odd, *swab*() copies and exchanges *nbytes*−1 bytes and the disposition of the last byte is unspecified. If copying takes place between objects that overlap, the behavior is undefined. If *nbytes* is negative, *swab*() does nothing.

RETURN VALUE

None.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

swprintf — print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int swprintf(wchar_t *restrict ws, size_t n,
   const wchar_t *restrict format, ...);
```

DESCRIPTION

Refer to *fwprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

swscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int swscanf(const wchar_t *restrict ws,
   const wchar_t *restrict format, ...);
```

DESCRIPTION

Refer to *fwscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

symlink, symlinkat — make a symbolic link

SYNOPSIS

#include <unistd.h>

int symlink(const char ***path1*, const char ***path2*);

#include <fcntl.h>

int symlinkat(const char ***path1*, int *fd*, const char ***path2*);

DESCRIPTION

The *symlink*() function shall create a symbolic link called *path2* that contains the string pointed to by *path1* (*path2* is the name of the symbolic link created, *path1* is the string contained in the symbolic link).

The string pointed to by *path1* shall be treated only as a string and shall not be validated as a pathname.

If the *symlink*() function fails for any reason other than **[EIO]**, any file named by *path2* shall be unaffected.

If *path2* names a symbolic link, *symlink*() shall fail and set *errno* to **[EEXIST]**.

The symbolic link's user ID shall be set to the process' effective user ID. The symbolic link's group ID shall be set to the group ID of the parent directory or to the effective group ID of the process. Implementations shall provide a way to initialize the symbolic link's group ID to the group ID of the parent directory. Implementations may, but need not, provide an implementation-defined way to initialize the symbolic link's group ID to the effective group ID of the calling process.

The values of the file mode bits for the created symbolic link are unspecified. All interfaces specified by POSIX.1-2008 shall behave as if the contents of symbolic links can always be read, except that the value of the file mode bits returned in the *st_mode* field of the **stat** structure is unspecified.

Upon successful completion, *symlink*() shall mark for update the last data access, last data modification, and last file status change timestamps of the symbolic link. Also, the last data modification and last file status change timestamps of the directory that contains the new entry shall be marked for update.

The *symlinkat*() function shall be equivalent to the *symlink*() function except in the case where *path2* specifies a relative path. In this case the symbolic link is created relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

If *symlinkat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *symlink*().

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error.

ERRORS

These functions shall fail if:

EACCES

Write permission is denied in the directory where the symbolic link is being created, or search permission is denied for a component of the path prefix of *path2*.

EEXIST

The *path2* argument names an existing file.

EIO An I/O error occurs while reading from or writing to the file system.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path2* argument.

ENAMETOOLONG
The length of a component of the pathname specified by the *path2* argument is longer than {NAME_MAX} or the length of the *path1* argument is longer than {SYMLINK_MAX}.

ENOENT
A component of the path prefix of *path2* does not name an existing file or *path2* is an empty string.

ENOENT or **ENOTDIR**
The *path2* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters. If *path2* without the trailing <slash> characters would name an existing file, an **[ENOENT]** error shall not occur.

ENOSPC
The directory in which the entry for the new symbolic link is being placed cannot be extended because no space is left on the file system containing the directory, or the new symbolic link cannot be created because no space is left on the file system which shall contain the link, or the file system is out of file-allocation resources.

ENOTDIR
A component of the path prefix of *path2* names an existing file that is neither a directory nor a symbolic link to a directory.

EROFS
The new symbolic link would reside on a read-only file system.

The *symlinkat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path2* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path2* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

These functions may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path2* argument.

ENAMETOOLONG
The length of the *path2* argument exceeds {PATH_MAX} or pathname resolution of a symbolic link in the *path2* argument produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES
None.

APPLICATION USAGE
Like a hard link, a symbolic link allows a file to have multiple logical names. The presence of a hard link guarantees the existence of a file, even after the original name has been removed. A symbolic link provides no such assurance; in fact, the file named by the *path1* argument need not exist when the link is created. A

symbolic link can cross file system boundaries.

Normal permission checks are made on each component of the symbolic link pathname during its resolution.

RATIONALE

The purpose of the *symlinkat*() function is to create symbolic links in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *symlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *symlinkat*() function it can be guaranteed that the created symbolic link is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir( ), *fstatat*( ), *lchown*( ), *link*( ), *open*( ), *readlink*( ), *rename*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sync — schedule file system updates

SYNOPSIS

```
#include <unistd.h>

void sync(void);
```

DESCRIPTION

The *sync*() function shall cause all information in memory that updates file systems to be scheduled for writing out to all file systems.

The writing, although scheduled, is not necessarily complete upon return from *sync*().

RETURN VALUE

The *sync*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fsync( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

sysconf — get configurable system variables

SYNOPSIS

#include <unistd.h>

long sysconf(int *name*);

DESCRIPTION

The *sysconf*() function provides a method for the application to determine the current value of a configurable system limit or option (*variable*). The implementation shall support all of the variables listed in the following table and may support others.

The *name* argument represents the system variable to be queried. The following table lists the minimal set of system variables from *<limits.h>* or *<unistd.h>* that can be returned by *sysconf*(), and the symbolic constants defined in *<unistd.h>* that are the corresponding values used for *name*.

box center tab(@); cB | cB lw(2.7i)1e | le. Variable@Value of Name _ {AIO_LISTIO_MAX}@_SC_AIO_LISTIO_MAX {AIO_MAX}@_SC_AIO_MAX {AIO_PRIO_DELTA_MAX}@_SC_AIO_PRIO_DELTA_MAX {ARG_MAX}@_SC_ARG_MAX {ATEXIT_MAX}@_SC_ATEXIT_MAX {BC_BASE_MAX}@_SC_BC_BASE_MAX {BC_DIM_MAX}@_SC_BC_DIM_MAX {BC_SCALE_MAX}@_SC_BC_SCALE_MAX {BC_STRING_MAX}@_SC_BC_STRING_MAX {CHILD_MAX}@_SC_CHILD_MAX Clock ticks/second@_SC_CLK_TCK {COLL_WEIGHTS_MAX}@_SC_COLL_WEIGHTS_MAX {DELAYTIMER_MAX}@_SC_DELAYTIMER_MAX {EXPR_NEST_MAX}@_SC_EXPR_NEST_MAX {HOST_NAME_MAX}@_SC_HOST_NAME_MAX {IOV_MAX}@_SC_IOV_MAX {LINE_MAX}@_SC_LINE_MAX {LOGIN_NAME_MAX}@_SC_LOGIN_NAME_MAX {NGROUPS_MAX}@_SC_NGROUPS_MAX Initial size of *getgrgid_r*( ) and@_SC_GETGR_R_SIZE_MAX *getgrnam_r*( ) data buffers Initial size of *getpwuid_r*( ) and@_SC_GETPW_R_SIZE_MAX *getpwnam_r*( ) data buffers {MQ_OPEN_MAX}@_SC_MQ_OPEN_MAX {MQ_PRIO_MAX}@_SC_MQ_PRIO_MAX {OPEN_MAX}@_SC_OPEN_MAX {PAGE_SIZE}@_SC_PAGE_SIZE {PAGESIZE}@_SC_PAGESIZE {PTHREAD_DESTRUCTOR_ITERATIONS}@_SC_THREAD_DESTRUCTOR_ITERATIONS {PTHREAD_KEYS_MAX}@_SC_THREAD_KEYS_MAX {PTHREAD_STACK_MIN}@_SC_THREAD_STACK_MIN {PTHREAD_THREADS_MAX}@_SC_THREAD_THREADS_MAX {RE_DUP_MAX}@_SC_RE_DUP_MAX {RTSIG_MAX}@_SC_RTSIG_MAX {SEM_NSEMS_MAX}@_SC_SEM_NSEMS_MAX {SEM_VALUE_MAX}@_SC_SEM_VALUE_MAX {SIGQUEUE_MAX}@_SC_SIGQUEUE_MAX {STREAM_MAX}@_SC_STREAM_MAX {SYMLOOP_MAX}@_SC_SYMLOOP_MAX {TIMER_MAX}@_SC_TIMER_MAX {TTY_NAME_MAX}@_SC_TTY_NAME_MAX {TZNAME_MAX}@_SC_TZNAME_MAX _POSIX_ADVISORY_INFO@_SC_ADVISORY_INFO _POSIX_BARRIERS@_SC_BARRIERS _POSIX_ASYNCHRONOUS_IO@_SC_ASYNCHRONOUS_IO _POSIX_CLOCK_SELECTION@_SC_CLOCK_SELECTION _POSIX_CPUTIME@_SC_CPUTIME _POSIX_FSYNC@_SC_FSYNC _POSIX_IPV6@_SC_IPV6 _POSIX_JOB_CONTROL@_SC_JOB_CONTROL _POSIX_MAPPED_FILES@_SC_MAPPED_FILES _POSIX_MEMLOCK@_SC_MEMLOCK _POSIX_MEMLOCK_RANGE@_SC_MEMLOCK_RANGE _POSIX_MEMORY_PROTECTION@_SC_MEMORY_PROTECTION _POSIX_MESSAGE_PASSING@_SC_MESSAGE_PASSING _POSIX_MONOTONIC_CLOCK@_SC_MONOTONIC_CLOCK _POSIX_PRIORITIZED_IO@_SC_PRIORITIZED_IO _POSIX_PRIORITY_SCHEDULING@_SC_PRIORITY_SCHEDULING _POSIX_RAW_SOCKETS@_SC_RAW_SOCKETS _POSIX_READER_WRITER_LOCKS@_SC_READER_WRITER_LOCKS _POSIX_REALTIME_SIGNALS@_SC_REALTIME_SIGNALS _POSIX_REGEXP@_SC_REGEXP

_POSIX_SAVED_IDS@_SC_SAVED_IDS _POSIX_SEMAPHORES@_SC_SEMAPHORES _POSIX_SHARED_MEMORY_OBJECTS@_SC_SHARED_MEMORY_OBJECTS _POSIX_SHELL@_SC_SHELL _POSIX_SPAWN@_SC_SPAWN _POSIX_SPIN_LOCKS@_SC_SPIN_LOCKS _POSIX_SPORADIC_SERVER@_SC_SPO- RADIC_SERVER _POSIX_SS_REPL_MAX@_SC_SS_REPL_MAX _POSIX_SYNCHRO- NIZED_IO@_SC_SYNCHRONIZED_IO _POSIX_THREAD_ATTR_STACK- ADDR@_SC_THREAD_ATTR_STACKADDR _POSIX_THREAD_ATTR_STACK- SIZE@_SC_THREAD_ATTR_STACKSIZE _POSIX_THREAD_CPUTIME@_SC_THREAD_CPUTIME _POSIX_THREAD_PRIO_IN- HERIT@_SC_THREAD_PRIO_INHERIT _POSIX_THREAD_PRIO_PRO- TECT@_SC_THREAD_PRIO_PROTECT _POSIX_THREAD_PRIORITY_SCHEDUL- ING@_SC_THREAD_PRIORITY_SCHEDULING _POSIX_THREAD_PROCESS_SHARED@_SC_THREAD_PROCESS_SHARED _POSIX_THREAD_ROBUST_PRIO_INHERIT@_SC_THREAD_ROBUST_PRIO_INHERIT _POSIX_THREAD_ROBUST_PRIO_PROTECT@_SC_THREAD_ROBUST_PRIO_PROTECT _POSIX_THREAD_SAFE_FUNCTIONS@_SC_THREAD_SAFE_FUNCTIONS _POSIX_THREAD_SPORADIC_SERVER@_SC_THREAD_SPORADIC_SERVER _POSIX_THREADS@_SC_THREADS _POSIX_TIMEOUTS@_SC_TIMEOUTS

box center tab(@); cB | cB lw(2.85i)1e | le. Variable@Value of Name _ _POSIX_TIMERS@_SC_TIMERS _POSIX_TRACE@_SC_TRACE _POSIX_TRACE_EVENT_FIL- TER@_SC_TRACE_EVENT_FILTER _POSIX_TRACE_EVENT_NAME_MAX@_SC_TRACE_EVENT_NAME_MAX _POSIX_TRACE_IN- HERIT@_SC_TRACE_INHERIT _POSIX_TRACE_LOG@_SC_TRACE_LOG _POSIX_TRACE_NAME_MAX@_SC_TRACE_NAME_MAX _POSIX_TRACE_SYS_MAX@_SC_TRACE_SYS_MAX _POSIX_TRACE_USER_EVENT_MAX@_SC_TRACE_USER_EVENT_MAX _POSIX_TYPED_MEMORY_OBJECTS@_SC_TYPED_MEMORY_OBJECTS _POSIX_VER- SION@_SC_VERSION _POSIX_V7_ILP32_OFF32@_SC_V7_ILP32_OFF32 _POSIX_V7_ILP32_OFFBIG@_SC_V7_ILP32_OFFBIG _POSIX_V7_LP64_OFF64@_SC_V7_LP64_OFF64 _POSIX_V7_LPBIG_OFFBIG@_SC_V7_LP- BIG_OFFBIG _POSIX_V6_ILP32_OFF32@_SC_V6_ILP32_OFF32 _POSIX_V6_ILP32_OFF- BIG@_SC_V6_ILP32_OFFBIG _POSIX_V6_LP64_OFF64@_SC_V6_LP64_OFF64 _POSIX_V6_LP- BIG_OFFBIG@_SC_V6_LPBIG_OFFBIG _POSIX2_C_BIND@_SC_2_C_BIND _POSIX2_C_DEV@_SC_2_C_DEV _POSIX2_CHAR_TERM@_SC_2_CHAR_TERM _POSIX2_FORT_DEV@_SC_2_FORT_DEV _POSIX2_FORT_RUN@_SC_2_FORT_RUN _POSIX2_LOCALEDEF@_SC_2_LOCALEDEF _POSIX2_PBS@_SC_2_PBS _POSIX2_PBS_AC- COUNTING@_SC_2_PBS_ACCOUNTING _POSIX2_PBS_CHECKPOINT@_SC_2_PBS_CHECK- POINT _POSIX2_PBS_LOCATE@_SC_2_PBS_LOCATE _POSIX2_PBS_MES- SAGE@_SC_2_PBS_MESSAGE _POSIX2_PBS_TRACK@_SC_2_PBS_TRACK _POSIX2_SW_DEV@_SC_2_SW_DEV _POSIX2_UPE@_SC_2_UPE _POSIX2_VER- SION@_SC_2_VERSION _XOPEN_CRYPT@_SC_XOPEN_CRYPT _XOPEN_ENH_I18N@_SC_XOPEN_ENH_I18N _XOPEN_REALTIME@_SC_XOPEN_REALTIME _XOPEN_REALTIME_THREADS@_SC_XOPEN_REALTIME_THREADS _XOPEN_SHM@_SC_XOPEN_SHM _XOPEN_STREAMS@_SC_XOPEN_STREAMS _XOPEN_UNIX@_SC_XOPEN_UNIX _XOPEN_UUCP@_SC_XOPEN_UUCP _XOPEN_VER- SION@_SC_XOPEN_VERSION

RETURN VALUE

If *name* is an invalid value, *sysconf*() shall return −1 and set *errno* to indicate the error. If the variable corresponding to *name* is described in *<limits.h>* as a maximum or minimum value and the variable has no limit, *sysconf*() shall return −1 without changing the value of *errno*. Note that indefinite limits do not imply infinite limits; see *<limits.h>*.

Otherwise, *sysconf*() shall return the current variable value on the system. The value returned shall not be more restrictive than the corresponding value described to the application when it was compiled with the implementation's *<limits.h>* or *<unistd.h>*. The value shall not change during the lifetime of the calling process, except that *sysconf*(_SC_OPEN_MAX) may return different values before and after a call to *setrlimit*() which changes the RLIMIT_NOFILE soft limit.

If the variable corresponding to *name* is dependent on an unsupported option, the results are unspecified.

ERRORS

The *sysconf*() function shall fail if:

EINVAL
The value of the *name* argument is invalid.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

As −1 is a permissible return value in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *sysconf*(), and, if it returns −1, check to see if *errno* is non-zero.

Application developers should check whether an option, such as _POSIX_TRACE, is supported prior to obtaining and using values for related variables, such as _POSIX_TRACE_NAME_MAX.

RATIONALE

This functionality was added in response to requirements of application developers and of system vendors who deal with many international system configurations. It is closely related to *pathconf*() and *fpathconf*().

Although a conforming application can run on all systems by never demanding more resources than the minimum values published in this volume of POSIX.1-2017, it is useful for that application to be able to use the actual value for the quantity of a resource available on any given system. To do this, the application makes use of the value of a symbolic constant in *<limits.h>* or *<unistd.h>*.

However, once compiled, the application must still be able to cope if the amount of resource available is increased. To that end, an application may need a means of determining the quantity of a resource, or the presence of an option, at execution time.

Two examples are offered:

1. Applications may wish to act differently on systems with or without job control. Applications vendors who wish to distribute only a single binary package to all instances of a computer architecture would be forced to assume job control is never available if it were to rely solely on the *<unistd.h>* value published in this volume of POSIX.1-2017.
2. International applications vendors occasionally require knowledge of the number of clock ticks per second. Without these facilities, they would be required to either distribute their applications partially in source form or to have 50 Hz and 60 Hz versions for the various countries in which they operate.

It is the knowledge that many applications are actually distributed widely in executable form that leads to this facility. If limited to the most restrictive values in the headers, such applications would have to be prepared to accept the most limited environments offered by the smallest microcomputers. Although this is entirely portable, there was a consensus that they should be able to take advantage of the facilities offered by large systems, without the restrictions associated with source and object distributions.

During the discussions of this feature, it was pointed out that it is almost always possible for an application to discern what a value might be at runtime by suitably testing the various functions themselves. And, in any event, it could always be written to adequately deal with error returns from the various functions. In the end, it was felt that this imposed an unreasonable level of complication and sophistication on the application developer.

This runtime facility is not meant to provide ever-changing values that applications have to check multiple times. The values are seen as changing no more frequently than once per system initialization, such as by a system administrator or operator with an automatic configuration program. This volume of POSIX.1-2017 specifies that they shall not change within the lifetime of the process.

Some values apply to the system overall and others vary at the file system or directory level. The latter are described in *fpathconf*( ).

Note that all values returned must be expressible as integers. String values were considered, but the additional flexibility of this approach was rejected due to its added complexity of implementation and use.

Some values, such as {PATH_MAX}, are sometimes so large that they must not be used to, say, allocate arrays. The *sysconf*() function returns a negative value to show that this symbolic constant is not even defined in this case.

Similar to *pathconf*(), this permits the implementation not to have a limit. When one resource is infinite, returning an error indicating that some other resource limit has been reached is conforming behavior.

FUTURE DIRECTIONS

None.

SEE ALSO

confstr( ), *fpathconf*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<unistd.h>**

The Shell and Utilities volume of POSIX.1-2017, *getconf*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

syslog — log a message

SYNOPSIS

#include <syslog.h>

void syslog(int *priority*, const char **message*, ... /* *argument* */);

DESCRIPTION

Refer to *closelog*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

system — issue a command

SYNOPSIS

```
#include <stdlib.h>

int system(const char *command);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *command* is a null pointer, the *system*() function shall determine whether the host environment has a command processor. If *command* is not a null pointer, the *system*() function shall pass the string pointed to by *command* to that command processor to be executed in an implementation-defined manner; this might then cause the program calling *system*() to behave in a non-conforming manner or to terminate.

The *system*() function shall behave as if a child process were created using *fork*(), and the child process invoked the *sh* utility using *execl*() as follows:

```
execl(<shell path>, "sh", "-c", command, (char *)0);
```

where <*shell path*> is an unspecified pathname for the *sh* utility. It is unspecified whether the handlers registered with *pthread_atfork*() are called as part of the creation of the child process.

The *system*() function shall ignore the SIGINT and SIGQUIT signals, and shall block the SIGCHLD signal, while waiting for the command to terminate. If this might cause the application to miss a signal that would have killed it, then the application should examine the return value from *system*() and take whatever action is appropriate to the application if the command terminated due to receipt of a signal.

The *system*() function shall not affect the termination status of any child of the calling processes other than the process or processes it itself creates.

The *system*() function shall not return until the child process has terminated.

The *system*() function need not be thread-safe.

RETURN VALUE

If *command* is a null pointer, *system*() shall return non-zero to indicate that a command processor is available, or zero if none is available. The *system*() function shall always return non-zero when *command* is NULL.

If *command* is not a null pointer, *system*() shall return the termination status of the command language interpreter in the format specified by *waitpid*(). The termination status shall be as defined for the *sh* utility; otherwise, the termination status is unspecified. If some error prevents the command language interpreter from executing after the child process is created, the return value from *system*() shall be as if the command language interpreter had terminated using *exit*(127) or *_exit*(127). If a child process cannot be created, or if the termination status for the command language interpreter cannot be obtained, *system*() shall return −1 and set *errno* to indicate the error.

ERRORS

The *system*() function may set *errno* values as described by *fork*( ).

In addition, *system*() may fail if:

ECHILD
The status of the child process created by *system*() is no longer available.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the return value of *system*() is not −1, its value can be decoded through the use of the macros described in <*sys/wait.h*>. For convenience, these macros are also provided in <*stdlib.h*>.

Note that, while *system*() must ignore SIGINT and SIGQUIT and block SIGCHLD while waiting for the child to terminate, the handling of signals in the executed command is as specified by *fork*() and *exec*. For example, if SIGINT is being caught or is set to SIG_DFL when *system*() is called, then the child is started with SIGINT handling set to SIG_DFL.

Ignoring SIGINT and SIGQUIT in the parent process prevents coordination problems (two processes reading from the same terminal, for example) when the executed command ignores or catches one of the signals. It is also usually the correct action when the user has given a command to the application to be executed synchronously (as in the **'!'** command in many interactive applications). In either case, the signal should be delivered only to the child process, not to the application itself. There is one situation where ignoring the signals might have less than the desired effect. This is when the application uses *system*() to perform some task invisible to the user. If the user typed the interrupt character (**"^C"**, for example) while *system*() is being used in this way, one would expect the application to be killed, but only the executed command is killed. Applications that use *system*() in this way should carefully check the return status from *system*() to see if the executed command was successful, and should take appropriate action when the command fails.

Blocking SIGCHLD while waiting for the child to terminate prevents the application from catching the signal and obtaining status from *system*()'s child process before *system*() can get the status itself.

The context in which the utility is ultimately executed may differ from that in which *system*() was called. For example, file descriptors that have the FD_CLOEXEC flag set are closed, and the process ID and parent process ID are different. Also, if the executed utility changes its environment variables or its current working directory, that change is not reflected in the caller's context.

There is no defined way for an application to find the specific path for the shell. However, *confstr*() can provide a value for *PATH* that is guaranteed to find the *sh* utility.

Using the *system*() function in more than one thread in a process or when the SIGCHLD signal is being manipulated by more than one thread in a process may produce unexpected results.

RATIONALE

The *system*() function should not be used by programs that have set user (or group) ID privileges. The *fork*() and *exec* family of functions (except *execlp*() and *execvp*()), should be used instead. This prevents any unforeseen manipulation of the environment of the user that could cause execution of commands not anticipated by the calling program.

There are three levels of specification for the *system*() function. The ISO C standard gives the most basic. It requires that the function exists, and defines a way for an application to query whether a command language interpreter exists. It says nothing about the command language or the environment in which the command is interpreted.

POSIX.1-2008 places additional restrictions on *system*(). It requires that if there is a command language interpreter, the environment must be as specified by *fork*() and *exec*. This ensures, for example, that close-on-*exec* works, that file locks are not inherited, and that the process ID is different. It also specifies the return value from *system*() when the command line can be run, thus giving the application some information about the command's completion status.

Finally, POSIX.1-2008 requires the command to be interpreted as in the shell command language defined in the Shell and Utilities volume of POSIX.1-2017.

Note that, *system*(NULL) is required to return non-zero, indicating that there is a command language interpreter. At first glance, this would seem to conflict with the ISO C standard which allows *system*(NULL) to return zero. There is no conflict, however. A system must have a command language interpreter, and is non-conforming if none is present. It is therefore permissible for the *system*() function on such a system to implement the behavior specified by the ISO C standard as long as it is understood that the implementation does not conform to POSIX.1-2008 if *system*(NULL) returns zero.

It was explicitly decided that when *command* is NULL, *system*() should not be required to check to make sure that the command language interpreter actually exists with the correct mode, that there are enough processes to execute it, and so on. The call *system*(NULL) could, theoretically, check for such problems as too many existing child processes, and return zero. However, it would be inappropriate to return zero due to such a (presumably) transient condition. If some condition exists that is not under the control of this application and that would cause any *system*() call to fail, that system has been rendered non-conforming.

Early drafts required, or allowed, *system*() to return with *errno* set to **[EINTR]** if it was interrupted with a signal. This error return was removed, and a requirement that *system*() not return until the child has terminated was added. This means that if a *waitpid*() call in *system*() exits with *errno* set to **[EINTR]**, *system*() must reissue the *waitpid*(). This change was made for two reasons:

1. There is no way for an application to clean up if *system*() returns **[EINTR]**, short of calling *wait*(), and that could have the undesirable effect of returning the status of children other than the one started by *system*().
2. While it might require a change in some historical implementations, those implementations already have to be changed because they use *wait*() instead of *waitpid*().

Note that if the application is catching SIGCHLD signals, it will receive such a signal before a successful *system*() call returns.

To conform to POSIX.1-2008, *system*() must use *waitpid*(), or some similar function, instead of *wait*().

The following code sample illustrates how *system*() might be implemented on an implementation conforming to POSIX.1-2008.

```
#include <signal.h>
int system(const char *cmd)
{
    int stat;
    pid_t pid;
    struct sigaction sa, savintr, savequit;
    sigset_t saveblock;
    if (cmd == NULL)
        return(1);
    sa.sa_handler = SIG_IGN;
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = 0;
    sigemptyset(&savintr.sa_mask);
    sigemptyset(&savequit.sa_mask);
    sigaction(SIGINT, &sa, &savintr);
    sigaction(SIGQUIT, &sa, &savequit);
    sigaddset(&sa.sa_mask, SIGCHLD);
    sigprocmask(SIG_BLOCK, &sa.sa_mask, &saveblock);
    if ((pid = fork()) == 0) {
        sigaction(SIGINT, &savintr, (struct sigaction *)0);
        sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
        sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
        execl("/bin/sh", "sh", "-c", cmd, (char *)0);
        _exit(127);
    }
    if (pid == -1) {
        stat = -1; /* errno comes from fork() */
    } else {
        while (waitpid(pid, &stat, 0) == -1) {
            if (errno != EINTR){
                stat = -1;
                break;
            }
        }
    }
    sigaction(SIGINT, &savintr, (struct sigaction *)0);
    sigaction(SIGQUIT, &savequit, (struct sigaction *)0);
    sigprocmask(SIG_SETMASK, &saveblock, (sigset_t *)0);
    return(stat);
}
```

Note that, while a particular implementation of *system*() (such as the one above) can assume a particular path for the shell, such a path is not necessarily valid on another system. The above example is not portable, and is not intended to be.

Note also that the above example implementation is not thread-safe. Implementations can provide a thread-safe *system*() function, but doing so involves complications such as how to restore the signal dispositions for SIGINT and SIGQUIT correctly if there are overlapping calls, and how to deal with cancellation. The example above would not restore the signal dispositions and would leak a process ID if cancelled. This does not matter for a non-thread-safe implementation since canceling a non-thread-safe function results in undefined behavior (see *Section 2.9.5.2*, *Cancellation Points*). To avoid leaking a process ID, a thread-safe implementation would need to terminate the child process when acting on a cancellation.

One reviewer suggested that an implementation of *system*() might want to use an environment variable such as *SHELL* to determine which command interpreter to use. The supposed implementation would use the default command interpreter if the one specified by the environment variable was not available. This would allow a user, when using an application that prompts for command lines to be processed using *system*(), to specify a different command interpreter. Such an implementation is discouraged. If the alternate command interpreter did not follow the command line syntax specified in the Shell and Utilities volume of POSIX.1-2017, then changing *SHELL* would render *system*() non-conforming. This would affect applications that expected the specified behavior from *system*(), and since the Shell and Utilities volume of POSIX.1-2017 does not mention that *SHELL* affects *system*(), the application would not know that it needed to unset *SHELL*.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.9.5.2, *Cancellation Points*, *exec*, *pipe*( ), *pthread_atfork*( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<signal.h>**, **<stdlib.h>**, **<sys_wait.h>**

The Shell and Utilities volume of POSIX.1-2017, *sh*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tan, tanf, tanl — tangent function

SYNOPSIS

#include <math.h>

double tan(double *x*);
float tanf(float *x*);
long double tanl(long double *x*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the tangent of their argument *x*, measured in radians.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the tangent of *x*.

If the correct value would cause underflow, and is not representable, a range error may occur, and *tan*(), *tanf*(), and *tanl*() shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *tan*(), *tanf*(), and *tanl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If *x* is ±Inf, a domain error shall occur, and either a NaN (if supported), or an implementation-defined value shall be returned.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

If the correct value would cause overflow, a range error shall occur and *tan*(), *tanf*(), and *tanl*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

ERRORS

These functions shall fail if:

Domain Error
The value of *x* is ±Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Range Error The result overflows

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Range Error The result underflows, or the value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

Taking the Tangent of a 45-Degree Angle

```
#include <math.h>
...
double radians = 45.0 * M_PI / 180;
double result;
...
result = tan (radians);
```

APPLICATION USAGE

There are no known floating-point representations such that for a normal argument, *tan*(*x*) is either overflow or underflow.

These functions may lose accuracy when their argument is near a multiple of $\pi/2$ or is far from 0.0.

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atan( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tanh, tanhf, tanhl — hyperbolic tangent functions

SYNOPSIS

```
#include <math.h>

double tanh(double x);
float tanhf(float x);
long double tanhl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute the hyperbolic tangent of their argument *x*.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the hyperbolic tangent of *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0, *x* shall be returned.

If *x* is ±Inf, ±1 shall be returned.

If *x* is subnormal, a range error may occur
and *x* should be returned.

If *x* is not returned, *tanh*(), *tanhf*(), and *tanhl*() shall return an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

ERRORS

These functions may fail if:

Range Error The value of *x* is subnormal.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

atanh( ), *feclearexcept*( ), *fetestexcept*( ), *isnan*( ), *tan*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tanl — tangent function

SYNOPSIS

```
#include <math.h>

long double tanl(long double x);
```

DESCRIPTION

Refer to *tan*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcdrain — wait for transmission of output

SYNOPSIS

#include <termios.h>

int tcdrain(int *fildes*);

DESCRIPTION

The *tcdrain*() function shall block until all output written to the object referred to by *fildes* is transmitted. The *fildes* argument is an open file descriptor associated with a terminal.

Any attempts to use *tcdrain*() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcdrain*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINTR
A signal interrupted *tcdrain*().

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcflush( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcflow — suspend or restart the transmission or reception of data

SYNOPSIS

```
#include <termios.h>

int tcflow(int fildes, int action);
```

DESCRIPTION

The *tcflow*() function shall suspend or restart transmission or reception of data on the object referred to by *fildes*, depending on the value of *action*. The *fildes* argument is an open file descriptor associated with a terminal.

- If *action* is TCOOFF, output shall be suspended.
- If *action* is TCOON, suspended output shall be restarted.
- If *action* is TCIOFF and *fildes* refers to a terminal device, the system shall transmit a STOP character, which is intended to cause the terminal device to stop transmitting data to the system. If *fildes* is associated with a pseudo-terminal, the STOP character need not be transmitted.
- If *action* is TCION and *fildes* refers to a terminal device, the system shall transmit a START character, which is intended to cause the terminal device to start transmitting data to the system. If *fildes* is associated with a pseudo-terminal, the START character need not be transmitted.

The default on the opening of a terminal file is that neither its input nor its output are suspended.

Attempts to use *tcflow*() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal, shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflow*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINVAL
The *action* argument is not a supported value.

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcsendbreak( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcflush — flush non-transmitted output data, non-read input data, or both

SYNOPSIS

```
#include <termios.h>

int tcflush(int fildes, int queue_selector);
```

DESCRIPTION

Upon successful completion, *tcflush*() shall discard data written to the object referred to by *fildes* (an open file descriptor associated with a terminal) but not transmitted, or data received but not read, depending on the value of *queue_selector*:

* If *queue_selector* is TCIFLUSH, it shall flush data received but not read.
* If *queue_selector* is TCOFLUSH, it shall flush data written but not transmitted.
* If *queue_selector* is TCIOFLUSH, it shall flush both data received but not read and data written but not transmitted.

Attempts to use *tcflush*() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcflush*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINVAL
The *queue_selector* argument is not a supported value.

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcdrain( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcgetattr — get the parameters associated with the terminal

SYNOPSIS

#include <termios.h>

int tcgetattr(int *fildes*, struct termios **termios_p*);

DESCRIPTION

The *tcgetattr*() function shall get the parameters associated with the terminal referred to by *fildes* and store them in the **termios** structure referenced by *termios_p*. The *fildes* argument is an open file descriptor associated with a terminal.

The *termios_p* argument is a pointer to a **termios** structure.

The *tcgetattr*() operation is allowed from any process.

If the terminal device supports different input and output baud rates, the baud rates stored in the **termios** structure returned by *tcgetattr*() shall reflect the actual baud rates, even if they are equal. If differing baud rates are not supported, the rate returned as the output baud rate shall be the actual baud rate. If the terminal device does not support split baud rates, the input baud rate stored in the **termios** structure shall be the output rate (as one of the symbolic values).

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetattr*() function shall fail if:

EBADF

The *fildes* argument is not a valid file descriptor.

ENOTTY

The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Care must be taken when changing the terminal attributes. Applications should always do a *tcgetattr*(), save the **termios** structure values returned, and then do a *tcsetattr*(), changing only the necessary fields. The application should use the values saved from the *tcgetattr*() to reset the terminal state whenever it is done with the terminal. This is necessary because terminal attributes apply to the underlying port and not to each individual open instance; that is, all processes that have used the terminal see the latest attribute changes.

A program that uses these functions should be written to catch all signals and take other appropriate actions to ensure that when the program terminates, whether planned or not, the terminal device's state is restored to its original state.

Existing practice dealing with error returns when only part of a request can be honored is based on calls to the *ioctl*() function. In historical BSD and System V implementations, the corresponding *ioctl*() returns zero if the requested actions were semantically correct, even if some of the requested changes could not be made. Many existing applications assume this behavior and would no longer work correctly if the return value were changed from zero to −1 in this case.

Note that either specification has a problem. When zero is returned, it implies everything succeeded even if some of the changes were not made. When −1 is returned, it implies everything failed even though some of the changes were made.

Applications that need all of the requested changes made to work properly should follow *tcsetattr*() with a call to *tcgetattr*() and compare the appropriate field values.

FUTURE DIRECTIONS

None.

SEE ALSO

tcsetattr( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcgetpgrp — get the foreground process group ID

SYNOPSIS

```
#include <unistd.h>

pid_t tcgetpgrp(int fildes);
```

DESCRIPTION

The *tcgetpgrp*() function shall return the value of the process group ID of the foreground process group associated with the terminal.

If there is no foreground process group, *tcgetpgrp*() shall return a value greater than 1 that does not match the process group ID of any existing process group.

The *tcgetpgrp*() function is allowed from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.

RETURN VALUE

Upon successful completion, *tcgetpgrp*() shall return the value of the process group ID of the foreground process associated with the terminal. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetpgrp*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

ENOTTY
The calling process does not have a controlling terminal, or the file is not the controlling terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setsid( ), *setpgid*( ), *tcsetpgrp*( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcgetsid — get the process group ID for the session leader for the controlling terminal

SYNOPSIS

#include <termios.h>

pid_t tcgetsid(int *fildes*);

DESCRIPTION

The *tcgetsid*() function shall obtain the process group ID of the session for which the terminal specified by *fildes* is the controlling terminal.

RETURN VALUE

Upon successful completion, *tcgetsid*() shall return the process group ID of the session associated with the terminal. Otherwise, a value of −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcgetsid*() function shall fail if:

EBADF

The *fildes* argument is not a valid file descriptor.

ENOTTY

The calling process does not have a controlling terminal, or the file is not the controlling terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcsendbreak — send a break for a specific duration

SYNOPSIS

#include <termios.h>

int tcsendbreak(int *fildes*, int *duration*);

DESCRIPTION

If the terminal is using asynchronous serial data transmission, *tcsendbreak*() shall cause transmission of a continuous stream of zero-valued bits for a specific duration. If *duration* is 0, it shall cause transmission of zero-valued bits for at least 0.25 seconds, and not more than 0.5 seconds. If *duration* is not 0, it shall send zero-valued bits for an implementation-defined period of time.

The *fildes* argument is an open file descriptor associated with a terminal.

If the terminal is not using asynchronous serial data transmission, it is implementation-defined whether *tcsendbreak*() sends data to generate a break condition or returns without taking any action.

Attempts to use *tcsendbreak*() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsendbreak*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcsetattr — set the parameters associated with the terminal

SYNOPSIS

```
#include <termios.h>

int tcsetattr(int fildes, int optional_actions,
   const struct termios *termios_p);
```

DESCRIPTION

The *tcsetattr*() function shall set the parameters associated with the terminal referred to by the open file descriptor *fildes* (an open file descriptor associated with a terminal) from the **termios** structure referenced by *termios_p* as follows:

* If *optional_actions* is TCSANOW, the change shall occur immediately.
* If *optional_actions* is TCSADRAIN, the change shall occur after all output written to *fildes* is transmitted. This function should be used when changing parameters that affect output.
* If *optional_actions* is TCSAFLUSH, the change shall occur after all output written to *fildes* is transmitted, and all input so far received but not read shall be discarded before the change is made.

If the output baud rate stored in the **termios** structure pointed to by *termios_p* is the zero baud rate, B0, the modem control lines shall no longer be asserted. Normally, this shall disconnect the line.

If the input baud rate stored in the **termios** structure pointed to by *termios_p* is 0, the input baud rate given to the hardware is the same as the output baud rate stored in the **termios** structure.

The *tcsetattr*() function shall return successfully if it was able to perform any of the requested actions, even if some of the requested actions could not be performed. It shall set all the attributes that the implementation supports as requested and leave all the attributes not supported by the implementation unchanged. If no part of the request can be honored, it shall return −1 and set *errno* to **[EINVAL]**. If the input and output baud rates differ and are a combination that is not supported, neither baud rate shall be changed. A subsequent call to *tcgetattr*() shall return the actual state of the terminal device (reflecting both the changes made and not made in the previous *tcsetattr*() call). The *tcsetattr*() function shall not change the values found in the **termios** structure under any circumstances.

The effect of *tcsetattr*() is undefined if the value of the **termios** structure pointed to by *termios_p* was not derived from the result of a call to *tcgetattr*() on *fildes*; an application should modify only fields and flags defined by this volume of POSIX.1-2017 between the call to *tcgetattr*() and *tcsetattr*(), leaving all other fields and flags unmodified.

No actions defined by this volume of POSIX.1-2017, other than a call to *tcsetattr*(), a close of the last file descriptor in the system associated with this terminal device, or an open of the first file descriptor in the system associated with this terminal device (using the O_TTY_INIT flag if it is non-zero and the device is not a pseudo-terminal), shall cause any of the terminal attributes defined by this volume of POSIX.1-2017 to change.

If *tcsetattr*() is called from a process which is a member of a background process group on a *fildes* associated with its controlling terminal:

* If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the operation completes normally and no signal is sent.
* Otherwise, a SIGTTOU signal shall be sent to the process group.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetattr*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINTR
A signal interrupted *tcsetattr*().

EINVAL
The *optional_actions* argument is not a supported value, or an attempt was made to change an attribute represented in the **termios** structure to an unsupported value.

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The file associated with *fildes* is not a terminal.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If trying to change baud rates, applications should call *tcsetattr*() then call *tcgetattr*() in order to determine what baud rates were actually selected.

In general, there are two reasons for an application to change the parameters associated with a terminal device:

1. The device already has working parameter settings but the application needs a different behavior, such as non-canonical mode instead of canonical mode. The application sets (or clears) only a few flags or *c_cc*[ ] values. Typically, the terminal device in this case is either the controlling terminal for the process or a pseudo-terminal.
2. The device is a modem or similar piece of equipment connected by a serial line, and it was not open before the application opened it. In this case, the application needs to initialize all of the parameter settings ``from scratch''. However, since the **termios** structure may include both standard and non-standard parameters, the application cannot just initialize the whole structure in an arbitrary way (e.g., using *memset*()) as this may cause some of the non-standard parameters to be set incorrectly, resulting in non-conforming behavior of the terminal device. Conversely, the application cannot just set the standard parameters, assuming that the non-standard parameters will already have suitable values, as the device might previously have been used with non-conforming parameter settings (and some implementations retain the settings from one use to the next). The solution is to open the terminal device using the O_TTY_INIT flag to initialize the terminal device to have conforming parameter settings, obtain those settings using *tcgetattr*(), and then set all of the standard parameters to the desired settings.

RATIONALE

The *tcsetattr*() function can be interrupted in the following situations:

* It is interrupted while waiting for output to drain.
* It is called from a process in a background process group and SIGTTOU is caught.

See also the RATIONALE section in *tcgetattr*( ).

FUTURE DIRECTIONS

Using an input baud rate of 0 to set the input rate equal to the output rate may not necessarily be supported in a future version of this volume of POSIX.1-2017.

SEE ALSO

cfgetispeed( ), *tcgetattr*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 11*, *General Terminal Interface*, **<termios.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tcsetpgrp — set the foreground process group ID

SYNOPSIS

#include <unistd.h>

int tcsetpgrp(int *fildes*, pid_t *pgid_id*);

DESCRIPTION

If the process has a controlling terminal, *tcsetpgrp*() shall set the foreground process group ID associated with the terminal to *pgid_id*. The application shall ensure that the file associated with *fildes* is the controlling terminal of the calling process and the controlling terminal is currently associated with the session of the calling process. The application shall ensure that the value of *pgid_id* matches a process group ID of a process in the same session as the calling process.

Attempts to use *tcsetpgrp*() from a process which is a member of a background process group on a *fildes* associated with its controlling terminal shall cause the process group to be sent a SIGTTOU signal. If the calling thread is blocking SIGTTOU signals or the process is ignoring SIGTTOU signals, the process shall be allowed to perform the operation, and no signal is sent.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *tcsetpgrp*() function shall fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

EINVAL
This implementation does not support the value in the *pgid_id* argument.

EIO The process group of the writing process is orphaned, the calling thread is not blocking SIGTTOU, and the process is not ignoring SIGTTOU.

ENOTTY
The calling process does not have a controlling terminal, or the file is not the controlling terminal, or the controlling terminal is no longer associated with the session of the calling process.

EPERM
The value of *pgid_id* is a value supported by the implementation, but does not match the process group ID of a process in the same session as the calling process.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

tcgetpgrp( )

The Base Definitions volume of POSIX.1-2017, **<sys_types.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tdelete, tfind, tsearch, twalk — manage a binary search tree

SYNOPSIS

```
#include <search.h>

void *tdelete(const void *restrict key, void **restrict rootp,
   int(*compar)(const void *, const void *));
void *tfind(const void *key, void *const *rootp,
   int(*compar)(const void *, const void *));
void *tsearch(const void *key, void **rootp,
   int (*compar)(const void *, const void *));
void twalk(const void *root,
   void (*action)(const void *, VISIT, int));
```

DESCRIPTION

The *tdelete*(), *tfind*(), *tsearch*(), and *twalk*() functions manipulate binary search trees. Comparisons are made with a user-supplied routine, the address of which is passed as the *compar* argument. This routine is called with two arguments, which are the pointers to the elements being compared. The application shall ensure that the user-supplied routine returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to, or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

The *tsearch*() function shall build and access the tree. The *key* argument is a pointer to an element to be accessed or stored. If there is a node in the tree whose element is equal to the value pointed to by *key*, a pointer to this found node shall be returned. Otherwise, the value pointed to by *key* shall be inserted (that is, a new node is created and the value of *key* is copied to this node), and a pointer to this node returned. Only pointers are copied, so the application shall ensure that the calling routine stores the data. The *rootp* argument points to a variable that points to the root node of the tree. A null pointer value for the variable pointed to by *rootp* denotes an empty tree; in this case, the variable shall be set to point to the node which shall be at the root of the new tree.

Like *tsearch*(), *tfind*() shall search for a node in the tree, returning a pointer to it if found. However, if it is not found, *tfind*() shall return a null pointer. The arguments for *tfind*() are the same as for *tsearch*().

The *tdelete*() function shall delete a node from a binary search tree. The arguments are the same as for *tsearch*(). The variable pointed to by *rootp* shall be changed if the deleted node was the root of the tree. If the deleted node was the root of the tree and had no children, the variable pointed to by *rootp* shall be set to a null pointer. The *tdelete*() function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

If *tsearch*() adds an element to a tree, or *tdelete*() successfully deletes an element from a tree, the concurrent use of that tree in another thread, or use of pointers produced by a previous call to *tfind*() or *tsearch*(), produces undefined results.

The *twalk*() function shall traverse a binary search tree. The *root* argument is a pointer to the root node of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) The argument *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument shall be the address of the node being visited. The structure pointed to by this argument is unspecified and shall not be modified by the application, but it shall be possible to cast a pointer-to-node into a pointer-to-pointer-to-element to access the element stored in the node. The second argument shall be a value from an enumeration data type:

```
typedef enum { preorder, postorder, endorder, leaf } VISIT;
```

(defined in <*search.h*>), depending on whether this is the first, second, or third time that the node is visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument shall be the level of the node in the tree, with the root being level 0.

If the calling function alters the pointer to the root, the result is undefined.

If the functions pointed to by *action* or *compar* (for any of these binary search functions) change the tree, the results are undefined.

These functions are thread-safe only as long as multiple threads do not access the same tree.

RETURN VALUE

If the node is found, both *tsearch*() and *tfind*() shall return a pointer to it. If not, *tfind*() shall return a null pointer, and *tsearch*() shall return a pointer to the inserted item.

A null pointer shall be returned by *tsearch*() if there is not enough space available to create a new node.

A null pointer shall be returned by *tdelete*(), *tfind*(), and *tsearch*() if *rootp* is a null pointer on entry.

The *tdelete*() function shall return a pointer to the parent of the deleted node, or an unspecified non-null pointer if the deleted node was the root node, or a null pointer if the node is not found.

The *twalk*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <limits.h>
#include <search.h>
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

struct element {      /* Pointers to these are stored in the tree. */
   int     count;
   char    string[];
};

void  *root = NULL;          /* This points to the root. */

int main(void)
{
   char   str[_POSIX2_LINE_MAX+1];
   int    length = 0;
   struct element *elementptr;
   void   *node;
   void   print_node(const void *, VISIT, int);
   int    node_compare(const void *, const void *),
          delete_root(const void *, const void *);

   while (fgets(str, sizeof(str), stdin))  {
      /* Set element. */
      length = strlen(str);
      if (str[length-1] == '\n')
         str[--length] = '\0';
```

```
        elementptr = malloc(sizeof(struct element) + length + 1);
        strcpy(elementptr->string, str);
        elementptr->count = 1;
        /* Put element into the tree. */
        node = tsearch((void *)elementptr, &root, node_compare);
        if (node == NULL) {
            fprintf(stderr,
                    "tsearch: Not enough space available\n");
            exit(EXIT_FAILURE);
        }
        else if (*(struct element **)node != elementptr) {
            /* A node containing the element already exists */
            (*(struct element **)node)->count++;
            free(elementptr);
        }
    }
    twalk(root, print_node);

    /* Delete all nodes in the tree */
    while (root != NULL) {
        elementptr = *(struct element **)root;
        printf("deleting node: string = %s,  count = %d\n",
               elementptr->string,
               elementptr->count);
        tdelete((void *)elementptr, &root, delete_root);
        free(elementptr);
    }

    return 0;
}

/*
 *  This routine compares two nodes, based on an
 *  alphabetical ordering of the string field.
 */
int
node_compare(const void *node1, const void *node2)
{
    return strcmp(((const struct element *) node1)->string,
        ((const struct element *) node2)->string);
}

/*
 *  This comparison routine can be used with tdelete()
 *  when explicitly deleting a root node, as no comparison
 *  is necessary.
 */
int
delete_root(const void *node1, const void *node2)
{
    return 0;
}

/*
 *  This routine prints out a node, the second time
 *  twalk encounters it or if it is a leaf.
 */
```

```
void
print_node(const void *ptr, VISIT order, int level)
{
    const struct element *p = *(const struct element **) ptr;

    if (order == postorder || order == leaf)  {
        (void) printf("string = %s,  count = %d\n",
            p->string, p->count);
    }
}
```

APPLICATION USAGE

The *root* argument to *twalk*() is one level of indirection less than the *rootp* arguments to *tdelete*() and *tsearch*().

There are two nomenclatures used to refer to the order in which tree nodes are visited. The *twalk*() function uses **preorder**, **postorder**, and **endorder** to refer respectively to visiting a node before any of its children, after its left child and before its right, and after both its children. The alternative nomenclature uses **preorder**, **inorder**, and **postorder** to refer to the same visits, which could result in some confusion over the meaning of **postorder**.

Since the return value of *tdelete*() is an unspecified non-null pointer in the case that the root of the tree has been deleted, applications should only use the return value of *tdelete*() as indication of success or failure and should not assume it can be dereferenced. Some implementations in this case will return a pointer to the new root of the tree (or to an empty tree if the deleted root node was the only node in the tree); other implementations return arbitrary non-null pointers.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

hcreate( ), *lsearch*( )

The Base Definitions volume of POSIX.1-2017, **<search.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

telldir — current location of a named directory stream

SYNOPSIS

#include <dirent.h>

long telldir(DIR **dirp*);

DESCRIPTION

The *telldir*() function shall obtain the current location associated with the directory stream specified by *dirp*.

If the most recent operation on the directory stream was a *seekdir*(), the directory position returned from the *telldir*() shall be the same as that supplied as a *loc* argument for *seekdir*().

RETURN VALUE

Upon successful completion, *telldir*() shall return the current location of the specified directory stream.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fdopendir( ), *readdir*( ), *seekdir*( )

The Base Definitions volume of POSIX.1-2017, **<dirent.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tempnam — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tempnam(const char *dir, const char *pfx);
```

DESCRIPTION

The *tempnam*() function shall generate a pathname that may be used for a temporary file.

The *tempnam*() function allows the user to control the choice of a directory. The *dir* argument points to the name of the directory in which the file is to be created. If *dir* is a null pointer or points to a string which is not a name for an appropriate directory, the path prefix defined as P_tmpdir in the *<stdio.h>* header shall be used. If that directory is not accessible, an implementation-defined directory may be used.

Many applications prefer their temporary files to have certain initial letter sequences in their names. The *pfx* argument should be used for this. This argument may be a null pointer or point to a string of up to five bytes to be used as the beginning of the filename.

Some implementations of *tempnam*() may use *tmpnam*() internally. On such implementations, if called more than {TMP_MAX} times in a single process, the behavior is implementation-defined.

RETURN VALUE

Upon successful completion, *tempnam*() shall allocate space for a string, put the generated pathname in that space, and return a pointer to it. The pointer shall be suitable for use in a subsequent call to *free*(). Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *tempnam*() function shall fail if:

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

Generating a Pathname

The following example generates a pathname for a temporary file in directory **/tmp**, with the prefix *file*. After the pathname has been created, the call to *free*() deallocates the space used to store the pathname.

```
#include <stdio.h>
#include <stdlib.h>
...
const char *directory = "/tmp";
const char *fileprefix = "file";
char *file;

file = tempnam(directory, fileprefix);
free(file);
```

APPLICATION USAGE

This function only creates pathnames. It is the application's responsibility to create and remove the files. Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile*() more useful.

Applications should use the *tmpfile*(), *mkdtemp*(), or *mkstemp*() functions instead of the obsolescent *tempnam*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *tempnam*() function may be removed in a future version.

SEE ALSO

fopen( ), *free*( ), *mkdtemp*( ), *open*( ), *tmpfile*( ), *tmpnam*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tfind — search binary search tree

SYNOPSIS

```
#include <search.h>

void *tfind(const void *key, void *const *rootp,
   int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *tdelete*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux. delim $$

NAME

tgamma, tgammaf, tgammal — compute gamma( ) function

SYNOPSIS

```
#include <math.h>

double tgamma(double x);
float tgammaf(float x);
long double tgammal(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall compute $Γ (^ x)$ where $Γ (^ x)$ is defined as $int from 0 to inf e"^" " "{ - t } t"^" " "{ x - 1 } dt$.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the gamma of *x*.

If *x* is a negative integer, a domain error may occur and either a NaN (if supported) or an implementation-defined value shall be returned. On systems that support the IEC 60559 Floating-Point option, a domain error shall occur and a NaN shall be returned.

If *x* is ±0, *tgamma*(), *tgammaf*(), and *tgammal*() shall return ±HUGE_VAL, ±HUGE_VALF, and ±HUGE_VALL, respectively. On systems that support the IEC 60559 Floating-Point option, a pole error shall occur; otherwise, a pole error may occur.

If the correct value would cause overflow, a range error shall occur and *tgamma*(), *tgammaf*(), and *tgammal*() shall return ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL, respectively, with the same sign as the correct value of the function.

If the correct value would cause underflow, and is not representable, a range error may occur, and *tgamma*(), *tgammaf*(), and *tgammal*() shall return 0.0, or (if IEC 60559 Floating-Point is not supported) an implementation-defined value no greater in magnitude than DBL_MIN, FLT_MIN, and LDBL_MIN, respectively.

If the correct value would cause underflow, and is representable, a range error may occur and the correct value shall be returned.

If *x* is subnormal and 1/*x* is representable, 1/*x* should be returned.

If *x* is NaN, a NaN shall be returned.

If *x* is +Inf, *x* shall be returned.

If *x* is −Inf, a domain error shall occur, and a NaN shall be returned.

ERRORS

These functions shall fail if:

Domain Error

The value of *x* is a negative integer, or *x* is −Inf.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno*

shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of *x* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error The value overflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

These functions may fail if:

Domain Error

The value of *x* is a negative integer.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error The value of *x* is zero.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error The result underflows.

If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

This function is named *tgamma*() in order to avoid conflicts with the historical *gamma*( ) and *lgamma*() functions.

FUTURE DIRECTIONS

It is possible that the error response for a negative integer argument may be changed to a pole error and a return value of ±Inf.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *lgamma*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

time — get time

SYNOPSIS

```
#include <time.h>

time_t time(time_t *tloc);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *time*() function shall return the value of time in seconds since the Epoch.

The *tloc* argument points to an area where the return value is also stored. If *tloc* is a null pointer, no value is stored.

RETURN VALUE

Upon successful completion, *time*() shall return the value of time. Otherwise, (**time_t**)−1 shall be returned.

ERRORS

The *time*() function may fail if:

EOVERFLOW
The number of seconds since the Epoch will not fit in an object of type **time_t**.

The following sections are informative.

EXAMPLES

Getting the Current Time

The following example uses the *time*() function to calculate the time elapsed, in seconds, since the Epoch, *localtime*() to convert that value to a broken-down time, and *asctime*() to convert the broken-down time values into a printable string.

```
#include <stdio.h>
#include <time.h>

int main(void)
{
time_t result;

    result = time(NULL);
    printf("%s%ju secs since the Epoch\n",
        asctime(localtime(&result)),
            (uintmax_t)result);
    return(0);
}
```

This example writes the current time to *stdout* in a form like this:

```
Wed Jun 26 10:32:15 1996
835810335 secs since the Epoch
```

Timing an Event

The following example gets the current time, prints it out in the user's format, and prints the number of minutes to an event being timed.

```
#include <time.h>
#include <stdio.h>
...
time_t now;
int minutes_to_event;
...
time(&now);
minutes_to_event = ...;
printf("The time is ");
puts(asctime(localtime(&now)));
printf("There are %d minutes to the event.\n",
    minutes_to_event);
...
```

APPLICATION USAGE

None.

RATIONALE

The *time*() function returns a value in seconds while *clock_gettime*() and *gettimeofday*() return a **struct timespec** (seconds and nanoseconds) and **struct timeval** (seconds and microseconds), respectively, and are therefore capable of returning more precise times. The *times*() function is also capable of more precision than *time*() as it returns a value in clock ticks, although it returns the elapsed time since an arbitrary point such as system boot time, not since the epoch.

Implementations in which **time_t** is a 32-bit signed integer (many historical implementations) fail in the year 2038. POSIX.1-2008 does not address this problem. However, the use of the **time_t** type is mandated in order to ease the eventual fix.

On some systems the *time*() function is implemented using a system call that does not return an error condition in addition to the return value. On these systems it is impossible to differentiate between valid and invalid return values and hence overflow conditions cannot be reliably detected.

The use of the *<time.h>* header instead of *<sys/types.h>* allows compatibility with the ISO C standard.

Many historical implementations (including Version 7) and the 1984 /usr/group standard use **long** instead of **time_t**. This volume of POSIX.1-2017 uses the latter type in order to agree with the ISO C standard.

FUTURE DIRECTIONS

In a future version of this volume of POSIX.1-2017, **time_t** is likely to be required to be capable of representing times far in the future. Whether this will be mandated as a 64-bit type or a requirement that a specific date in the future be representable (for example, 10000 AD) is not yet determined. Systems purchased after the approval of this volume of POSIX.1-2017 should be evaluated to determine whether their lifetime will extend past 2038.

SEE ALSO

asctime( ), *clock*( ), *clock_getres*( ), *ctime*( ), *difftime*( ), *futimens*( ), *gettimeofday*( ), *gmtime*( ), *localtime*( ), *mktime*( ), *strftime*( ), *strptime*( ), *times*( ), *utime*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

timer_create — create a per-process timer

SYNOPSIS

```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clockid, struct sigevent *restrict evp,
   timer_t *restrict timerid);
```

DESCRIPTION

The *timer_create*() function shall create a per-process timer using the specified clock, *clock_id*, as the timing base. The *timer_create*() function shall return, in the location referenced by *timerid*, a timer ID of type **timer_t** used to identify the timer in timer requests. This timer ID shall be unique within the calling process until the timer is deleted. The particular clock, *clock_id*, is defined in *<time.h>*. The timer whose ID is returned shall be in a disarmed state upon return from *timer_create*().

The *evp* argument, if non-NULL, points to a **sigevent** structure. This structure, allocated by the application, defines the asynchronous notification to occur as specified in *Section 2.4.1*, *Signal Generation and Delivery* when the timer expires. If the *evp* argument is NULL, the effect is as if the *evp* argument pointed to a **sigevent** structure with the *sigev_notify* member having the value SIGEV_SIGNAL, the *sigev_signo* having a default signal number, and the *sigev_value* member having the value of the timer ID.

Each implementation shall define a set of clocks that can be used as timing bases for per-process timers. All implementations shall support a *clock_id* of CLOCK_REALTIME. If the Monotonic Clock option is supported, implementations shall support a *clock_id* of CLOCK_MONOTONIC.

Per-process timers shall not be inherited by a child process across a *fork*() and shall be disarmed and deleted by an *exec*.

If _POSIX_CPUTIME is defined, implementations shall support *clock_id* values representing the CPU-time clock of the calling process.

If _POSIX_THREAD_CPUTIME is defined, implementations shall support *clock_id* values representing the CPU-time clock of the calling thread.

It is implementation-defined whether a *timer_create*() function will succeed if the value defined by *clock_id* corresponds to the CPU-time clock of a process or thread different from the process or thread invoking the function.

If *evp*−>*sigev_sigev_notify* is SIGEV_THREAD and *sev*−>*sigev_notify_attributes* is not NULL, if the attribute pointed to by *sev*−>*sigev_notify_attributes* has a thread stack address specified by a call to *pthread_attr_setstack*(), the results are unspecified if the signal is generated more than once.

RETURN VALUE

If the call succeeds, *timer_create*() shall return zero and update the location referenced by *timerid* to a **timer_t**, which can be passed to the per-process timer calls. If an error occurs, the function shall return a value of −1 and set *errno* to indicate the error. The value of *timerid* is undefined if an error occurs.

ERRORS

The *timer_create*() function shall fail if:

EAGAIN
The system lacks sufficient signal queuing resources to honor the request.

EAGAIN
The calling process has already created all of the timers it is allowed by this implementation.

EINVAL
The specified clock ID is not defined.

ENOTSUP
The implementation does not support the creation of a timer attached to the CPU-time clock that is specified by *clock_id* and associated with a process or thread different from the process or thread invoking *timer_create*().

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If a timer is created which has *evp−>sigev_sigev_notify* set to SIGEV_THREAD and the attribute pointed to by *evp−>sigev_notify_attributes* has a thread stack address specified by a call to *pthread_attr_setstack*(), the memory dedicated as a thread stack cannot be recovered. The reason for this is that the threads created in response to a timer expiration are created detached, or in an unspecified way if the thread attribute's *detachstate* is PTHREAD_CREATE_JOINABLE. In neither case is it valid to call *pthread_join*(), which makes it impossible to determine the lifetime of the created thread which thus means the stack memory cannot be reused.

RATIONALE

Periodic Timer Overrun and Resource Allocation

The specified timer facilities may deliver realtime signals (that is, queued signals) on implementations that support this option. Since realtime applications cannot afford to lose notifications of asynchronous events, like timer expirations or asynchronous I/O completions, it must be possible to ensure that sufficient resources exist to deliver the signal when the event occurs. In general, this is not a difficulty because there is a one-to-one correspondence between a request and a subsequent signal generation. If the request cannot allocate the signal delivery resources, it can fail the call with an **[EAGAIN]** error.

Periodic timers are a special case. A single request can generate an unspecified number of signals. This is not a problem if the requesting process can service the signals as fast as they are generated, thus making the signal delivery resources available for delivery of subsequent periodic timer expiration signals. But, in general, this cannot be assured—processing of periodic timer signals may ''overrun''; that is, subsequent periodic timer expirations may occur before the currently pending signal has been delivered.

Also, for signals, according to the POSIX.1-1990 standard, if subsequent occurrences of a pending signal are generated, it is implementation-defined whether a signal is delivered for each occurrence. This is not adequate for some realtime applications. So a mechanism is required to allow applications to detect how many timer expirations were delayed without requiring an indefinite amount of system resources to store the delayed expirations.

The specified facilities provide for an overrun count. The overrun count is defined as the number of extra timer expirations that occurred between the time a timer expiration signal is generated and the time the signal is delivered. The signal-catching function, if it is concerned with overruns, can retrieve this count on entry. With this method, a periodic timer only needs one ''signal queuing resource'' that can be allocated at the time of the *timer_create*() function call.

A function is defined to retrieve the overrun count so that an application need not allocate static storage to contain the count, and an implementation need not update this storage asynchronously on timer expirations. But, for some high-frequency periodic applications, the overhead of an additional system call on each timer expiration may be prohibitive. The functions, as defined, permit an implementation to maintain the overrun count in user space, associated with the *timerid*. The *timer_getoverrun*() function can then be implemented as a macro that uses the *timerid* argument (which may just be a pointer to a user space structure containing the counter) to locate the overrun count with no system call overhead. Other implementations, less concerned with this class of applications, can avoid the asynchronous update of user space by maintaining the count in a system structure at the cost of the extra system call to obtain it.

Timer Expiration Signal Parameters

The Realtime Signals Extension option supports an application-specific datum that is delivered to the extended signal handler. This value is explicitly specified by the application, along with the signal number to be delivered, in a **sigevent** structure. The type of the application-defined value can be either an integer constant or a pointer. This explicit specification of the value, as opposed to always sending the timer ID, was selected based on existing practice.

It is common practice for realtime applications (on non-POSIX systems or realtime extended POSIX systems) to use the parameters of event handlers as the case label of a switch statement or as a pointer to an application-defined data structure. Since *timer_id*s are dynamically allocated by the *timer_create*() function, they can be used for neither of these functions without additional application overhead in the signal handler; for example, to search an array of saved timer IDs to associate the ID with a constant or application data structure.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *timer_delete*( ), *timer_getoverrun*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

timer_delete — delete a per-process timer

SYNOPSIS

```
#include <time.h>

int timer_delete(timer_t timerid);
```

DESCRIPTION

The *timer_delete*() function deletes the specified timer, *timerid*, previously created by the *timer_create*() function. If the timer is armed when *timer_delete*() is called, the behavior shall be as if the timer is automatically disarmed before removal. The disposition of pending signals for the deleted timer is unspecified.

The behavior is undefined if the value specified by the *timerid* argument to *timer_delete*() does not correspond to a timer ID returned by *timer_create*() but not yet deleted by *timer_delete*().

RETURN VALUE

If successful, the *timer_delete*() function shall return a value of zero. Otherwise, the function shall return a value of −1 and set *errno* to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

If an implementation detects that the value specified by the *timerid* argument to *timer_delete*() does not correspond to a timer ID returned by *timer_create*() but not yet deleted by *timer_delete*(), it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

timer_create( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

timer_getoverrun, timer_gettime, timer_settime — per-process timers

SYNOPSIS

```
#include <time.h>

int timer_getoverrun(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
    const struct itimerspec *restrict value,
    struct itimerspec *restrict ovalue);
```

DESCRIPTION

The *timer_gettime*() function shall store the amount of time until the specified timer, *timerid*, expires and the reload value of the timer into the space pointed to by the *value* argument. The *it_value* member of this structure shall contain the amount of time before the timer expires, or zero if the timer is disarmed. This value is returned as the interval until timer expiration, even if the timer was armed with absolute time. The *it_interval* member of *value* shall contain the reload value last set by *timer_settime*().

The *timer_settime*() function shall set the time until the next expiration of the timer specified by *timerid* from the *it_value* member of the *value* argument and arm the timer if the *it_value* member of *value* is non-zero. If the specified timer was already armed when *timer_settime*() is called, this call shall reset the time until next expiration to the *value* specified. If the *it_value* member of *value* is zero, the timer shall be disarmed. The effect of disarming or resetting a timer with pending expiration notifications is unspecified.

If the flag TIMER_ABSTIME is not set in the argument *flags*, *timer_settime*() shall behave as if the time until next expiration is set to be equal to the interval specified by the *it_value* member of *value*. That is, the timer shall expire in *it_value* nanoseconds from when the call is made. If the flag TIMER_ABSTIME is set in the argument *flags*, *timer_settime*() shall behave as if the time until next expiration is set to be equal to the difference between the absolute time specified by the *it_value* member of *value* and the current value of the clock associated with *timerid*. That is, the timer shall expire when the clock reaches the value specified by the *it_value* member of *value*. If the specified time has already passed, the function shall succeed and the expiration notification shall be made.

The reload value of the timer shall be set to the value specified by the *it_interval* member of *value*. When a timer is armed with a non-zero *it_interval*, a periodic (or repetitive) timer is specified.

Time values that are between two consecutive non-negative integer multiples of the resolution of the specified timer shall be rounded up to the larger multiple of the resolution. Quantization error shall not cause the timer to expire earlier than the rounded time value.

If the argument *ovalue* is not NULL, the *timer_settime*() function shall store, in the location referenced by *ovalue*, a value representing the previous amount of time before the timer would have expired, or zero if the timer was disarmed, together with the previous timer reload value. Timers shall not expire before their scheduled time.

Only a single signal shall be queued to the process for a given timer at any point in time. When a timer for which a signal is still pending expires, no signal shall be queued, and a timer overrun shall occur. When a timer expiration signal is delivered to or accepted by a process, the *timer_getoverrun*() function shall return the timer expiration overrun count for the specified timer. The overrun count returned contains the number of extra timer expirations that occurred between the time the signal was generated (queued) and when it was delivered or accepted, up to but not including an implementation-defined maximum of {DELAYTIMER_MAX}. If the number of such extra expirations is greater than or equal to {DELAYTIMER_MAX}, then the overrun count shall be set to {DELAYTIMER_MAX}. The value returned by *timer_getoverrun*() shall apply to the most recent expiration signal delivery or acceptance for the timer. If no expiration signal has been delivered for the timer, the return value of *timer_getoverrun*() is unspecified.

The behavior is undefined if the value specified by the *timerid* argument to *timer_getoverrun*(), *timer_gettime*(), or *timer_settime*() does not correspond to a timer ID returned by *timer_create*() but not yet deleted by *timer_delete*().

RETURN VALUE

If the *timer_getoverrun*() function succeeds, it shall return the timer expiration overrun count as explained above.

If the *timer_gettime*() or *timer_settime*() functions succeed, a value of 0 shall be returned.

If an error occurs for any of these functions, the value −1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *timer_settime*() function shall fail if:

EINVAL
A *value* structure specified a nanosecond value less than zero or greater than or equal to 1 000 million, and the *it_value* member of that structure did not specify zero seconds and nanoseconds.

The *timer_settime*() function may fail if:

EINVAL
The *it_interval* member of *value* is not zero and the timer was created with notification by creation of a new thread (*sigev_sigev_notify* was SIGEV_THREAD) and a fixed stack address has been set in the thread attribute pointed to by *sigev_notify_attributes*.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Using fixed stack addresses is problematic when timer expiration is signaled by the creation of a new thread. Since it cannot be assumed that the thread created for one expiration is finished before the next expiration of the timer, it could happen that two threads use the same memory as a stack at the same time. This is invalid and produces undefined results.

RATIONALE

Practical clocks tick at a finite rate, with rates of 100 hertz and 1 000 hertz being common. The inverse of this tick rate is the clock resolution, also called the clock granularity, which in either case is expressed as a time duration, being 10 milliseconds and 1 millisecond respectively for these common rates. The granularity of practical clocks implies that if one reads a given clock twice in rapid succession, one may get the same time value twice; and that timers must wait for the next clock tick after the theoretical expiration time, to ensure that a timer never returns too soon. Note also that the granularity of the clock may be significantly coarser than the resolution of the data format used to set and get time and interval values. Also note that some implementations may choose to adjust time and/or interval values to exactly match the ticks of the underlying clock.

This volume of POSIX.1-2017 defines functions that allow an application to determine the implementation-supported resolution for the clocks and requires an implementation to document the resolution supported for timers and *nanosleep*() if they differ from the supported clock resolution. This is more of a procurement issue than a runtime application issue.

If an implementation detects that the value specified by the *timerid* argument to *timer_getoverrun*(), *timer_gettime*(), or *timer_settime*() does not correspond to a timer ID returned by *timer_create*() but not yet deleted by *timer_delete*(), it is recommended that the function should fail and report an **[EINVAL]** error.

FUTURE DIRECTIONS

None.

SEE ALSO

clock_getres( ), *timer_create*( )

The Base Definitions volume of POSIX.1-2017, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

times — get process and waited-for child process times

SYNOPSIS

```
#include <sys/times.h>

clock_t times(struct tms *buffer);
```

DESCRIPTION

The *times*() function shall fill the **tms** structure pointed to by *buffer* with time-accounting information. The **tms** structure is defined in *<sys/times.h>*.

All times are measured in terms of the number of clock ticks used.

The times of a terminated child process shall be included in the *tms_cutime* and *tms_cstime* elements of the parent when *wait*(), *waitid*(), or *waitpid*() returns the process ID of this terminated child. If a child process has not waited for its children, their times shall not be included in its times.

* The *tms_utime* structure member is the CPU time charged for the execution of user instructions of the calling process.
* The *tms_stime* structure member is the CPU time charged for execution by the system on behalf of the calling process.
* The *tms_cutime* structure member is the sum of the *tms_utime* and *tms_cutime* times of the child processes.
* The *tms_cstime* structure member is the sum of the *tms_stime* and *tms_cstime* times of the child processes.

RETURN VALUE

Upon successful completion, *times*() shall return the elapsed real time, in clock ticks, since an arbitrary point in the past (for example, system start-up time). This point does not change from one invocation of *times*() within the process to another. The return value may overflow the possible range of type **clock_t**. If *times*() fails, (**clock_t**)−1 shall be returned and *errno* set to indicate the error.

ERRORS

The *times*() function shall fail if:

EOVERFLOW
The return value would overflow the range of **clock_t**.

The following sections are informative.

EXAMPLES

Timing a Database Lookup

The following example defines two functions, *start_clock*() and *end_clock*(), that are used to time a lookup. It also defines variables of type **clock_t** and **tms** to measure the duration of transactions. The *start_clock*() function saves the beginning times given by the *times*() function. The *end_clock*() function gets the ending times and prints the difference between the two times.

```
#include <sys/times.h>
#include <stdio.h>
...
void start_clock(void);
void end_clock(char *msg);
...
```

```
static clock_t st_time;
static clock_t en_time;
static struct tms st_cpu;
static struct tms en_cpu;
...
void
start_clock()
{
    st_time = times(&st_cpu);
}

/* This example assumes that the result of each subtraction
   is within the range of values that can be represented in
   an integer type. */
void
end_clock(char *msg)
{
    en_time = times(&en_cpu);

    fputs(msg,stdout);
    printf("Real Time: %jd, User Time %jd, System Time %jd\n",
        (intmax_t)(en_time - st_time),
        (intmax_t)(en_cpu.tms_utime - st_cpu.tms_utime),
        (intmax_t)(en_cpu.tms_stime - st_cpu.tms_stime));
}
```

APPLICATION USAGE

Applications should use *sysconf*(_SC_CLK_TCK) to determine the number of clock ticks per second as it may vary from system to system.

RATIONALE

The accuracy of the times reported is intentionally left unspecified to allow implementations flexibility in design, from uniprocessor to multi-processor networks.

The inclusion of times of child processes is recursive, so that a parent process may collect the total times of all of its descendants. But the times of a child are only added to those of its parent when its parent successfully waits on the child. Thus, it is not guaranteed that a parent process can always see the total times of all its descendants; see also the discussion of the term ''realtime'' in *alarm*( ).

If the type **clock_t** is defined to be a signed 32-bit integer, it overflows in somewhat more than a year if there are 60 clock ticks per second, or less than a year if there are 100. There are individual systems that run continuously for longer than that. This volume of POSIX.1-2017 permits an implementation to make the reference point for the returned value be the start-up time of the process, rather than system start-up time.

The term ''charge'' in this context has nothing to do with billing for services. The operating system accounts for time used in this way. That information must be correct, regardless of how that information is used.

FUTURE DIRECTIONS

None.

SEE ALSO

alarm( ), *exec*, *fork*( ), *sysconf*( ), *time*( ), *wait*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, **<sys_times.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers,

Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

timezone — difference from UTC and local standard time

SYNOPSIS

```
#include <time.h>

extern long timezone;
```

DESCRIPTION

Refer to *tzset*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tmpfile — create a temporary file

SYNOPSIS

```
#include <stdio.h>

FILE *tmpfile(void);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *tmpfile*() function shall create a temporary file and open a corresponding stream. The file shall be automatically deleted when all references to the file are closed. The file shall be opened as in *fopen*() for update (*wb*+), except that implementations may restrict the permissions, either by clearing the file mode bits or setting them to the value S_IRUSR | S_IWUSR.

In some implementations, a permanent file may be left behind if the process calling *tmpfile*() is killed while it is processing a call to *tmpfile*().

An error message may be written to standard error if the stream cannot be opened.

RETURN VALUE

Upon successful completion, *tmpfile*() shall return a pointer to the stream of the file that is created. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *tmpfile*() function shall fail if:

EINTR
A signal was caught during *tmpfile*().

EMFILE
All file descriptors available to the process are currently open.

EMFILE
{STREAM_MAX} streams are currently open in the calling process.

ENFILE
The maximum allowable number of files is currently open in the system.

ENOSPC
The directory or file system which would contain the new file cannot be expanded.

EOVERFLOW
The file is a regular file and the size of the file cannot be represented correctly in an object of type **off_t**.

The *tmpfile*() function may fail if:

EMFILE
{FOPEN_MAX} streams are currently open in the calling process.

ENOMEM
Insufficient storage space is available.

The following sections are informative.

EXAMPLES

Creating a Temporary File

The following example creates a temporary file for update, and returns a pointer to a stream for the created file in the *fp* variable.

```
#include <stdio.h>
...
FILE *fp;

fp = tmpfile ();
```

APPLICATION USAGE

It should be possible to open at least {TMP_MAX} temporary files during the lifetime of the program (this limit may be shared with *tmpnam*()) and there should be no limit on the number simultaneously open other than this limit and any limit on the number of open file descriptors or streams ({OPEN_MAX}, {FOPEN_MAX}, {STREAM_MAX}).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fopen*( ), *mkdtemp*( ), *tmpnam*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tmpnam — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam(char *s);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *tmpnam*() function shall generate a string that is a valid pathname that does not name an existing file. The function is potentially capable of generating {TMP_MAX} different strings, but any or all of them may already be in use by existing files and thus not be suitable return values.

The *tmpnam*() function generates a different string each time it is called from the same process, up to {TMP_MAX} times. If it is called more than {TMP_MAX} times, the behavior is implementation-defined.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017, except *tempnam*(), calls *tmpnam*().

The *tmpnam*() function need not be thread-safe if called with a NULL parameter.

RETURN VALUE

Upon successful completion, *tmpnam*() shall return a pointer to a string. If no suitable string can be generated, the *tmpnam*() function shall return a null pointer.

If the argument *s* is a null pointer, *tmpnam*() shall leave its result in an internal static object and return a pointer to that object. Subsequent calls to *tmpnam*() may modify the same object. If the argument *s* is not a null pointer, it is presumed to point to an array of at least L_tmpnam **char**s; *tmpnam*() shall write its result in that array and shall return the argument as its value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Generating a Pathname

The following example generates a unique pathname and stores it in the array pointed to by *ptr*.

```
#include <stdio.h>
...
char pathname[L_tmpnam+1];
char *ptr;

ptr = tmpnam(pathname);
```

APPLICATION USAGE

This function only creates pathnames. It is the application's responsibility to create and remove the files.

Between the time a pathname is created and the file is opened, it is possible for some other process to create a file with the same name. Applications may find *tmpfile*() more useful.

Applications should use the *tmpfile*(), *mkstemp*(), or *mkdtemp*() functions instead of the obsolescent *tmpnam*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *tmpnam*() function may be removed in a future version.

SEE ALSO

fopen( ), *open*( ), *mkdtemp*( ), *tempnam*( ), *tmpfile*( ), *unlink*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

toascii — translate an integer to a 7-bit ASCII character

SYNOPSIS

#include <ctype.h>

int toascii(int *c*);

DESCRIPTION

The *toascii*() function shall convert its argument into a 7-bit ASCII character.

RETURN VALUE

The *toascii*() function shall return the value (*c* &0x7f).

ERRORS

No errors are returned.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *toascii*() function cannot be used portably in a localized application.

RATIONALE

None.

FUTURE DIRECTIONS

The *toascii*() function may be removed in a future version.

SEE ALSO

isascii( )

The Base Definitions volume of POSIX.1-2017, **<ctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tolower, tolower_l — transliterate uppercase characters to lowercase

SYNOPSIS

#include <ctype.h>

int tolower(int *c*);
int tolower_l(int *c*, locale_t *locale*);

DESCRIPTION

For *tolower*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *tolower*() and *tolower_l*() functions have as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined. If the argument of *tolower*() or *tolower_l*() represents an uppercase letter, and there exists a corresponding lowercase letter as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding lowercase letter. All other arguments in the domain are returned unchanged.

The behavior is undefined if the *locale* argument to *tolower_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, the *tolower*() and *tolower_l*() functions shall return the lowercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

toupper, toupper_l — transliterate lowercase characters to uppercase

SYNOPSIS

```
#include <ctype.h>

int toupper(int c);
int toupper_l(int c, locale_t locale);
```

DESCRIPTION

For *toupper*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *toupper*() and *toupper_l*() functions have as a domain a type **int**, the value of which is representable as an **unsigned char** or the value of EOF. If the argument has any other value, the behavior is undefined.

If the argument of *toupper*() or *toupper_l*() represents a lowercase letter, and there exists a corresponding uppercase letter as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding uppercase letter.

All other arguments in the domain are returned unchanged.

The behavior is undefined if the *locale* argument to *toupper_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *toupper*() and *toupper_l*() shall return the uppercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<ctype.h>**, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

towctrans, towctrans_l — wide-character transliteration

SYNOPSIS

```
#include <wctype.h>

wint_t towctrans(wint_t wc, wctrans_t desc);
wint_t towctrans_l(wint_t wc, wctrans_t desc,
   locale_t locale);
```

DESCRIPTION

For *towctrans*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *towctrans*() and *towctrans_l*() functions shall transliterate the wide-character code *wc* using the mapping described by *desc*.

The current setting of the *LC_CTYPE* category in the current locale or in the locale represented by *locale*, respectively, should be the same as during the call to *wctrans*() or *wctrans_l*() that returned the value *desc*.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans*() or *desc* is invalidated by a subsequent call to *setlocale*() that has affected category *LC_CTYPE*), the result is unspecified.

If the value of *desc* is invalid (that is, not obtained by a call to *wctrans_l*() with the same locale object *locale*) the result is unspecified.

An application wishing to check for error situations should set *errno* to 0 before calling *towctrans*() or *towctrans_l*().

If *errno* is non-zero on return, an error has occurred.

The behavior is undefined if the *locale* argument to *towctrans_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

If successful, the *towctrans*() and *towctrans_l*() functions shall return the mapped value of *wc* using the mapping described by *desc*. Otherwise, they shall return *wc* unchanged.

ERRORS

These functions may fail if:

EINVAL

desc contains an invalid transliteration descriptor.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The strings **"tolower"** and **"toupper"** are reserved for the standard mapping names. In the table below, the functions in the left column are equivalent to the functions in the right column.

towlower(*wc*)	towctrans(*wc*, wctrans("tolower"))
towlower_l(*wc*, *locale*)	towctrans_l(*wc*, wctrans("tolower"), *locale*)
towupper(*wc*)	towctrans(*wc*, wctrans("toupper"))
towupper_l(*wc*, *locale*)	towctrans_l(*wc*, wctrans("toupper"), *locale*)

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

towlower( ), *towupper*( ), *wctrans*( )

The Base Definitions volume of POSIX.1-2017, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

towlower, towlower_l — transliterate uppercase wide-character code to lowercase

SYNOPSIS

```
#include <wctype.h>

wint_t towlower(wint_t wc);
wint_t towlower_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *towlower*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *towlower*() and *towlower_l*() functions have as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the locale used by the function or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towlower*() or *towlower_l*() represents an uppercase wide-character code, and there exists a corresponding lowercase wide-character code as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding lowercase wide-character code. All other arguments in the domain are returned unchanged.

The behavior is undefined if the *locale* argument to *towlower_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, the *towlower*() and *towlower_l*() functions shall return the lowercase letter corresponding to the argument passed; otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

towupper, towupper_l — transliterate lowercase wide-character code to uppercase

SYNOPSIS

```
#include <wctype.h>

wint_t towupper(wint_t wc);
wint_t towupper_l(wint_t wc, locale_t locale);
```

DESCRIPTION

For *towupper*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *towupper*() and *towupper_l*() functions have as a domain a type **wint_t**, the value of which the application shall ensure is a character representable as a **wchar_t**, and a wide-character code corresponding to a valid character in the locale used by the function or the value of WEOF. If the argument has any other value, the behavior is undefined. If the argument of *towupper*() or *towupper_l*() represents a lowercase wide-character code, and there exists a corresponding uppercase wide-character code as defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*), the result shall be the corresponding uppercase wide-character code. All other arguments in the domain are returned unchanged.

The behavior is undefined if the *locale* argument to *towupper_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, the *towupper*() and *towupper_l*() functions shall return the uppercase letter corresponding to the argument passed. Otherwise, they shall return the argument unchanged.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

setlocale( ), *uselocale*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<locale.h>**, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

trunc, truncf, truncl — round to truncated integer value

SYNOPSIS

```
#include <math.h>

double trunc(double x);
float truncf(float x);
long double truncl(long double x);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall round their argument to the integer value, in floating format, nearest to but no larger in magnitude than the argument.

RETURN VALUE

Upon successful completion, these functions shall return the truncated integer value.
The result shall have the same sign as *x*.

If *x* is NaN, a NaN shall be returned.

If *x* is ±0 or ±Inf, *x* shall be returned.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The integral value returned by these functions need not be expressible as an **intmax_t**. The return value should be tested before assigning it to an integer type to avoid the undefined results of an integer overflow.

These functions may raise the inexact floating-point exception if the result differs in value from the argument.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

truncate — truncate a file to a specified length

SYNOPSIS

#include <unistd.h>

int truncate(const char **path*, off_t *length*);

DESCRIPTION

The *truncate*() function shall cause the regular file named by *path* to have a size which shall be equal to *length* bytes.

If the file previously was larger than *length*, the extra data is discarded. If the file was previously shorter than *length*, its size is increased, and the extended area appears as if it were zero-filled.

The application shall ensure that the process has write permission for the file.

If the request would cause the file size to exceed the soft file size limit for the process, the request shall fail and the implementation shall generate the SIGXFSZ signal for the process.

The *truncate*() function shall not modify the file offset for any open file descriptions associated with the file. Upon successful completion, *truncate*() shall mark for update the last data modification and last file status change timestamps of the file, and the S_ISUID and S_ISGID bits of the file mode may be cleared.

RETURN VALUE

Upon successful completion, *truncate*() shall return 0. Otherwise, −1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *truncate*() function shall fail if:

EINTR
A signal was caught during execution.

EINVAL
The *length* argument was less than 0.

EFBIG or **EINVAL**
The *length* argument was greater than the maximum file size.

EIO An I/O error occurred while reading from or writing to a file system.

EACCES
A component of the path prefix denies search permission, or write permission is denied on the file.

EISDIR
The named file is a directory.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file

that is neither a directory nor a symbolic link to a directory.

EROFS
The named file resides on a read-only file system.

The *truncate*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

open( )

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

truncf, truncl — round to truncated integer value

SYNOPSIS

```
#include <math.h>

float truncf(float x);
long double truncl(long double x);
```

DESCRIPTION

Refer to *trunc*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

tsearch — search a binary search tree

SYNOPSIS

```
#include <search.h>

void *tsearch(const void *key, void **rootp,
   int (*compar)(const void *, const void *));
```

DESCRIPTION

Refer to *tdelete*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ttyname, ttyname_r — find the pathname of a terminal

SYNOPSIS

#include <unistd.h>

char *ttyname(int *fildes*);
int ttyname_r(int *fildes*, char **name*, size_t *namesize*);

DESCRIPTION

The *ttyname*() function shall return a pointer to a string containing a null-terminated pathname of the terminal associated with file descriptor *fildes*. The application shall not modify the string returned. The returned pointer might be invalidated or the string content might be overwritten by a subsequent call to *ttyname*(). The returned pointer and the string content might also be invalidated if the calling thread is terminated.

The *ttyname*() function need not be thread-safe.

The *ttyname_r*() function shall store the null-terminated pathname of the terminal associated with the file descriptor *fildes* in the character array referenced by *name*. The array is *namesize* characters long and should have space for the name and the terminating null character. The maximum length of the terminal name shall be {TTY_NAME_MAX}.

RETURN VALUE

Upon successful completion, *ttyname*() shall return a pointer to a string. Otherwise, a null pointer shall be returned and *errno* set to indicate the error.

If successful, the *ttyname_r*() function shall return zero. Otherwise, an error number shall be returned to indicate the error.

ERRORS

The *ttyname*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

ENOTTY
The file associated with the *fildes* argument is not a terminal.

The *ttyname_r*() function may fail if:

EBADF
The *fildes* argument is not a valid file descriptor.

ENOTTY
The file associated with the *fildes* argument is not a terminal.

ERANGE
The value of *namesize* is smaller than the length of the string to be returned including the terminating null character.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

The term "terminal" is used instead of the historical term "terminal device" in order to avoid a reference to an undefined term.

The thread-safe version places the terminal name in a user-supplied buffer and returns a non-zero value if it fails. The non-thread-safe version may return the name in a static data area that may be overwritten by each call.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

twalk — traverse a binary search tree

SYNOPSIS

```
#include <search.h>

void twalk(const void *root,
    void (*action)(const void *, VISIT, int ));
```

DESCRIPTION

Refer to *tdelete*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

daylight, timezone, tzname, tzset — set timezone conversion information

SYNOPSIS

```
#include <time.h>

extern int daylight;
extern long timezone;
extern char *tzname[2];
void tzset(void);
```

DESCRIPTION

The *tzset*() function shall use the value of the environment variable *TZ* to set time conversion information used by *ctime*( ), *localtime*( ), *mktime*( ), and *strftime*( ). If *TZ* is absent from the environment, implementation-defined default timezone information shall be used.

The *tzset*() function shall set the external variable *tzname* as follows:

```
tzname[0] = "std";
tzname[1] = "dst";
```

where *std* and *dst* are as described in the Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*.

The *tzset*() function also shall set the external variable *daylight* to 0 if Daylight Savings Time conversions should never be applied for the timezone in use; otherwise, non-zero. The external variable *timezone* shall be set to the difference, in seconds, between Coordinated Universal Time (UTC) and local standard time.

If a thread accesses *tzname*, *daylight*, or *timezone* directly while another thread is in a call to *tzset*(), or to any function that is required or allowed to set timezone information as if by calling *tzset*(), the behavior is undefined.

RETURN VALUE

The *tzset*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

Example *TZ* variables and their timezone differences are given in the table below:

center box tab(!); cI | cI lw(1i) | lw(1i). TZ!timezone _ EST5EDT!5*60*60 GMT0!0*60*60 JST-9!−9*60*60 MET-1MEST!−1*60*60 MST7MDT!7*60*60 PST8PDT!8*60*60

APPLICATION USAGE

Since the *ctime*(), *localtime*(), *mktime*(), *strftime*(), and *strftime_l*() functions are required to set timezone information as if by calling *tzset*(), there is no need for an explicit *tzset*() call before using these functions. However, portable applications should call *tzset*() explicitly before using *ctime_r*() or *localtime_r*() because setting timezone information is optional for those functions.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

ctime( ), *localtime*( ), *mktime*( ), *strftime*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 8*, *Environment Variables*, **<time.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ulimit — get and set process limits

SYNOPSIS

#include <ulimit.h>

long ulimit(int *cmd*, ...);

DESCRIPTION

The *ulimit*() function shall control process limits. The process limits that can be controlled by this function include the maximum size of a single file that can be written (this is equivalent to using *setrlimit*() with RLIMIT_FSIZE). The *cmd* values, defined in *<ulimit.h>*, include:

UL_GETFSIZE

Return the file size limit (RLIMIT_FSIZE) of the process. The limit shall be in units of 512-byte blocks and shall be inherited by child processes. Files of any size can be read. The return value shall be the integer part of the soft file size limit divided by 512. If the result cannot be represented as a **long**, the result is unspecified.

UL_SETFSIZE

Set the file size limit for output operations of the process to the value of the second argument, taken as a **long**, multiplied by 512. If the result would overflow an **rlim_t**, the actual value set is unspecified. Any process may decrease its own limit, but only a process with appropriate privileges may increase the limit. The return value shall be the integer part of the new file size limit divided by 512.

The *ulimit*() function shall not change the setting of *errno* if successful.

As all return values are permissible in a successful situation, an application wishing to check for error situations should set *errno* to 0, then call *ulimit*(), and, if it returns −1, check to see if *errno* is non-zero.

RETURN VALUE

Upon successful completion, *ulimit*() shall return the value of the requested limit. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *ulimit*() function shall fail and the limit shall be unchanged if:

EINVAL

The *cmd* argument is not valid.

EPERM

A process not having appropriate privileges attempts to increase its file size limit.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since the *ulimit*() function uses type **long** rather than **rlim_t**, this function is not sufficient for file sizes on many current systems. Applications should use the *getrlimit*() or *setrlimit*() functions instead of the obsolescent *ulimit*() function.

RATIONALE

None.

FUTURE DIRECTIONS

The *ulimit*() function may be removed in a future version.

SEE ALSO

exec, *getrlimit*( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<ulimit.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

umask — set and get the file mode creation mask

SYNOPSIS

#include <sys/stat.h>

mode_t umask(mode_t *cmask*);

DESCRIPTION

The *umask*() function shall set the file mode creation mask of the process to *cmask* and return the previous value of the mask. Only the file permission bits of *cmask* (see *<sys/stat.h>*) are used; the meaning of the other bits is implementation-defined.

The file mode creation mask of the process is used to turn off permission bits in the *mode* argument supplied during calls to the following functions:

* *open*(), *openat*(), *creat*(), *mkdir*(), *mkdirat*(), *mkfifo*(), and *mkfifoat*()
* *mknod*(), *mknodat*()
* *mq_open*()
* *sem_open*()

Bit positions that are set in *cmask* are cleared in the mode of the created file.

RETURN VALUE

The file permission bits in the value returned by *umask*() shall be the previous value of the file mode creation mask. The state of any other bits in that value is unspecified, except that a subsequent call to *umask*() with the returned value as *cmask* shall leave the state of the mask the same as its state before the first call, including any unspecified use of those bits.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Unsigned argument and return types for *umask*() were proposed. The return type and the argument were both changed to **mode_t**.

Historical implementations have made use of additional bits in *cmask* for their implementation-defined purposes. The addition of the text that the meaning of other bits of the field is implementation-defined permits these implementations to conform to this volume of POSIX.1-2017.

FUTURE DIRECTIONS

None.

SEE ALSO

creat( ), *exec*, *mkdir*( ), *mkfifo*( ), *mknod*( ), *mq_open*( ), *open*( ), *sem_open*( )

The Base Definitions volume of POSIX.1-2017, **<sys_stat.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

uname — get the name of the current system

SYNOPSIS

#include <sys/utsname.h>

int uname(struct utsname **name*);

DESCRIPTION

The *uname*() function shall store information identifying the current system in the structure pointed to by *name*.

The *uname*() function uses the **utsname** structure defined in *<sys/utsname.h>*.

The *uname*() function shall return a string naming the current system in the character array *sysname*. Similarly, *nodename* shall contain the name of this node within an implementation-defined communications network. The arrays *release* and *version* shall further identify the operating system. The array *machine* shall contain a name that identifies the hardware that the system is running on.

The format of each member is implementation-defined.

RETURN VALUE

Upon successful completion, a non-negative value shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The inclusion of the *nodename* member in this structure does not imply that it is sufficient information for interfacing to communications networks.

RATIONALE

The values of the structure members are not constrained to have any relation to the version of this volume of POSIX.1-2017 implemented in the operating system. An application should instead depend on _POSIX_VERSION and related constants defined in *<unistd.h>*.

This volume of POSIX.1-2017 does not define the sizes of the members of the structure and permits them to be of different sizes, although most implementations define them all to be the same size: eight bytes plus one byte for the string terminator. That size for *nodename* is not enough for use with many networks.

The *uname*() function originated in System III, System V, and related implementations, and it does not exist in Version 7 or 4.3 BSD. The values it returns are set at system compile time in those historical implementations.

4.3 BSD has *gethostname*() and *gethostid*(), which return a symbolic name and a numeric value, respectively. There are related *sethostname*() and *sethostid*() functions that are used to set the values the other two functions return. The former functions are included in this specification, the latter are not.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<sys_utsname.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ungetc — push byte back into input stream

SYNOPSIS

#include <stdio.h>

int ungetc(int *c*, FILE **stream*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ungetc*() function shall push the byte specified by *c* (converted to an **unsigned char**) back onto the input stream pointed to by *stream*. The pushed-back bytes shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*(), *fseeko*(), *fsetpos*(), or *rewind*()) or *fflush*() shall discard any pushed-back bytes for the stream. The external storage corresponding to the stream shall be unchanged.

One byte of push-back shall be provided. If *ungetc*() is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *c* equals that of the macro EOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetc*() shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after all pushed-back bytes have been read, or discarded by calling *fseek*(), *fseeko*(), *fsetpos*(), or *rewind*() (but not *fflush*()), shall be the same as it was before the bytes were pushed back. The file-position indicator is decremented by each successful call to *ungetc*(); if its value was 0 before a call, its value is unspecified after the call.

RETURN VALUE

Upon successful completion, *ungetc*() shall return the byte pushed back after conversion. Otherwise, it shall return EOF.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fseek*( ), *getc*( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

ungetwc — push wide-character code back into the input stream

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

wint_t ungetwc(wint_t wc, FILE *stream);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *ungetwc*() function shall push the character corresponding to the wide-character code specified by *wc* back onto the input stream pointed to by *stream*. The pushed-back characters shall be returned by subsequent reads on that stream in the reverse order of their pushing. A successful intervening call (with the stream pointed to by *stream*) to a file-positioning function (*fseek*(), *fseeko*(), *fsetpos*(), or *rewind*()) or *fflush*() shall discard any pushed-back characters for the stream. The external storage corresponding to the stream is unchanged.

At least one character of push-back shall be provided. If *ungetwc*() is called too many times on the same stream without an intervening read or file-positioning operation on that stream, the operation may fail.

If the value of *wc* equals that of the macro WEOF, the operation shall fail and the input stream shall be left unchanged.

A successful call to *ungetwc*() shall clear the end-of-file indicator for the stream. The value of the file-position indicator for the stream after all pushed-back characters have been read, or discarded by calling *fseek*(), *fseeko*(), *fsetpos*(), or *rewind*() (but not *fflush*()), shall be the same as it was before the characters were pushed back. The file-position indicator is decremented (by one or more) by each successful call to *ungetwc*(); if its value was 0 before a call, its value is unspecified after the call.

RETURN VALUE

Upon successful completion, *ungetwc*() shall return the wide-character code corresponding to the pushed-back character. Otherwise, it shall return WEOF.

ERRORS

The *ungetwc*() function may fail if:

EILSEQ
An invalid character sequence is detected, or a wide-character code does not correspond to a valid character.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fseek*( ), *fsetpos*( ), *read*( ), *rewind*( ), *setbuf*( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

unlink, unlinkat — remove a directory entry

SYNOPSIS

```
#include <unistd.h>

int unlink(const char *path);

#include <fcntl.h>

int unlinkat(int fd, const char *path, int flag);
```

DESCRIPTION

The *unlink*() function shall remove a link to a file. If *path* names a symbolic link, *unlink*() shall remove the symbolic link named by *path* and shall not affect any file or directory named by the contents of the symbolic link. Otherwise, *unlink*() shall remove the link named by the pathname pointed to by *path* and shall decrement the link count of the file referenced by the link.

When the file's link count becomes 0 and no process has the file open, the space occupied by the file shall be freed and the file shall no longer be accessible. If one or more processes have the file open when the last link is removed, the link shall be removed before *unlink*() returns, but the removal of the file contents shall be postponed until all references to the file are closed.

The *path* argument shall not name a directory unless the process has appropriate privileges and the implementation supports using *unlink*() on directories.

Upon successful completion, *unlink*() shall mark for update the last data modification and last file status change timestamps of the parent directory. Also, if the file's link count is not 0, the last file status change timestamp of the file shall be marked for update.

The *unlinkat*() function shall be equivalent to the *unlink*() or *rmdir*() function except in the case where *path* specifies a relative path. In this case the directory entry to be removed is determined relative to the directory associated with the file descriptor *fd* instead of the current working directory. If the access mode of the open file description associated with the file descriptor is not O_SEARCH, the function shall check whether directory searches are permitted using the current permissions of the directory underlying the file descriptor. If the access mode is O_SEARCH, the function shall not perform the check.

Values for *flag* are constructed by a bitwise-inclusive OR of flags from the following list, defined in *<fcntl.h>*:

AT_REMOVEDIR
: Remove the directory entry specified by *fd* and *path* as a directory, not a normal file.

If *unlinkat*() is passed the special value AT_FDCWD in the *fd* parameter, the current working directory shall be used and the behavior shall be identical to a call to *unlink*() or *rmdir*() respectively, depending on whether or not the AT_REMOVEDIR bit is set in *flag*.

RETURN VALUE

Upon successful completion, these functions shall return 0. Otherwise, these functions shall return −1 and set *errno* to indicate the error. If −1 is returned, the named file shall not be changed.

ERRORS

These functions shall fail and shall not unlink the file if:

EACCES
: Search permission is denied for a component of the path prefix, or write permission is denied on the directory containing the directory entry to be removed.

EBUSY
The file named by the *path* argument cannot be unlinked because it is being used by the system or another process and the implementation considers this an error.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The file named by *path* is a directory, and either the calling process does not have appropriate privileges, or the implementation prohibits using *unlink*() on directories.

EPERM or **EACCES**
The S_ISVTX flag is set on the directory containing the file referred to by the *path* argument and the process does not satisfy the criteria specified in the Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*.

EROFS
The directory entry to be unlinked is part of a read-only file system.

The *unlinkat*() function shall fail if:

EACCES
The access mode of the open file description associated with *fd* is not O_SEARCH and the permissions of the directory underlying *fd* do not permit directory searches.

EBADF
The *path* argument does not specify an absolute path and the *fd* argument is neither AT_FDCWD nor a valid file descriptor open for reading or searching.

ENOTDIR
The *path* argument is not an absolute path and *fd* is a file descriptor associated with a non-directory file.

EEXIST or **ENOTEMPTY**
The *flag* parameter has the AT_REMOVEDIR bit set and the *path* argument names a directory that is not an empty directory, or there are hard links to the directory other than dot or a single entry in dot-dot.

ENOTDIR
The *flag* parameter has the AT_REMOVEDIR bit set and *path* does not name a directory.

These functions may fail and not unlink the file if:

EBUSY
The file named by *path* is a named STREAM.

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

ETXTBSY
The entry to be unlinked is the last directory entry to a pure procedure (shared text) file that is being executed.

The *unlinkat*() function may fail if:

EINVAL
The value of the *flag* argument is not valid.

The following sections are informative.

EXAMPLES

Removing a Link to a File

The following example shows how to remove a link to a file named **/home/cnd/mod1** by removing the entry named **/modules/pass1**.

```
#include <unistd.h>

char *path = "/modules/pass1";
int   status;
...
status = unlink(path);
```

Checking for an Error

The following example fragment creates a temporary password lock file named **LOCKFILE**, which is defined as **/etc/ptmp**, and gets a file descriptor for it. If the file cannot be opened for writing, *unlink*() is used to remove the link between the file descriptor and **LOCKFILE**.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"

int pfd;  /* Integer for file descriptor returned by open call. */
FILE *fpfd;  /* File pointer for use in putpwent(). */
...
/* Open password Lock file. If it exists, this is an error. */
if ((pfd = open(LOCKFILE, O_WRONLY| O_CREAT | O_EXCL, S_IRUSR
    | S_IWUSR | S_IRGRP | S_IROTH)) == -1)  {
    fprintf(stderr, "Cannot open /etc/ptmp. Try again later.\n");
    exit(1);
}

/* Lock file created; proceed with fdopen of lock file so that
   putpwent() can be used.
 */
if ((fpfd = fdopen(pfd, "w")) == NULL) {
    close(pfd);
    unlink(LOCKFILE);
    exit(1);
}
```

Replacing Files

The following example fragment uses *unlink*() to discard links to files, so that they can be replaced with new versions of the files. The first call removes the link to **LOCKFILE** if an error occurs. Successive calls remove the links to **SAVEFILE** and **PASSWDFILE** so that new links can be created, then removes the link to **LOCKFILE** when it is no longer needed.

```
#include <sys/types.h>
#include <stdio.h>
#include <fcntl.h>
#include <errno.h>
#include <unistd.h>
#include <sys/stat.h>

#define LOCKFILE "/etc/ptmp"
#define PASSWDFILE "/etc/passwd"
#define SAVEFILE "/etc/opasswd"
...
/* If no change was made, assume error and leave passwd unchanged. */
if (!valid_change) {
   fprintf(stderr, "Could not change password for user %s\n", user);
   unlink(LOCKFILE);
   exit(1);
}

/* Change permissions on new password file. */
chmod(LOCKFILE, S_IRUSR | S_IRGRP | S_IROTH);

/* Remove saved password file. */
unlink(SAVEFILE);

/* Save current password file. */
link(PASSWDFILE, SAVEFILE);

/* Remove current password file. */
unlink(PASSWDFILE);

/* Save new password file as current password file. */
link(LOCKFILE,PASSWDFILE);

/* Remove lock file. */
unlink(LOCKFILE);

exit(0);
```

APPLICATION USAGE

Applications should use *rmdir*() to remove a directory.

RATIONALE

Unlinking a directory is restricted to the superuser in many historical implementations for reasons given in *link*() (see also *rename*()).

The meaning of **[EBUSY]** in historical implementations is "mount point busy". Since this volume of POSIX.1-2017 does not cover the system administration concepts of mounting and unmounting, the description of the error was changed to "resource busy". (This meaning is used by some device drivers when a second process tries to open an exclusive use device.) The wording is also intended to allow implementations to refuse to remove a directory if it is the root or current working directory of any process.

The standard developers reviewed TR 24715-2006 and noted that LSB-conforming implementations may return **[EISDIR]** instead of **[EPERM]** when unlinking a directory. A change to permit this behavior by changing the requirement for **[EPERM]** to **[EPERM]** or **[EISDIR]** was considered, but decided against since it would break existing strictly conforming and conforming applications. Applications written for portability to both POSIX.1-2008 and the LSB should be prepared to handle either error code.

The purpose of the *unlinkat*() function is to remove directory entries in directories other than the current working directory without exposure to race conditions. Any part of the path of a file could be changed in parallel to a call to *unlink*(), resulting in unspecified behavior. By opening a file descriptor for the target directory and using the *unlinkat*() function it can be guaranteed that the removed directory entry is located relative to the desired directory.

FUTURE DIRECTIONS

None.

SEE ALSO

close( ), *link*( ), *remove*( ), *rename*( ), *rmdir*( ), *symlink*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.3*, *Directory Protection*, **<fcntl.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

unlockpt — unlock a pseudo-terminal master/slave pair

SYNOPSIS

```
#include <stdlib.h>

int unlockpt(int fildes);
```

DESCRIPTION

The *unlockpt*() function shall unlock the slave pseudo-terminal device associated with the master to which *fildes* refers.

Conforming applications shall ensure that they call *unlockpt*() before opening the slave side of a pseudo-terminal device.

RETURN VALUE

Upon successful completion, *unlockpt*() shall return 0. Otherwise, it shall return −1 and set *errno* to indicate the error.

ERRORS

The *unlockpt*() function may fail if:

EBADF
: The *fildes* argument is not a file descriptor open for writing.

EINVAL
: The *fildes* argument is not associated with a master pseudo-terminal device.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

See the RATIONALE section for *posix_openpt*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

grantpt( ), *open*( ), *posix_openpt*( ), *ptsname*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

unsetenv — remove an environment variable

SYNOPSIS

```
#include <stdlib.h>

int unsetenv(const char *name);
```

DESCRIPTION

The *unsetenv*() function shall remove an environment variable from the environment of the calling process. The *name* argument points to a string, which is the name of the variable to be removed. The named argument shall not contain an **'='** character. If the named variable does not exist in the current environment, the environment shall be unchanged and the function is considered to have completed successfully.

The *unsetenv*() function shall update the list of pointers to which *environ* points.

The *unsetenv*() function need not be thread-safe.

RETURN VALUE

Upon successful completion, zero shall be returned. Otherwise, −1 shall be returned, *errno* set to indicate the error, and the environment shall be unchanged.

ERRORS

The *unsetenv*() function shall fail if:

EINVAL
The *name* argument points to an empty string, or points to a string containing an **'='** character.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

Refer to the RATIONALE section in *setenv*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

getenv( ), *setenv*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**, **<sys_types.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

uselocale — use locale in current thread

SYNOPSIS

```
#include <locale.h>

locale_t uselocale(locale_t newloc);
```

DESCRIPTION

The *uselocale*() function shall set or query the current locale for the calling thread.

The value for the *newloc* argument shall be one of the following:

1. A value returned by the *newlocale*() or *duplocale*() functions
2. The special locale object descriptor LC_GLOBAL_LOCALE
3. (**locale_t**)0

If the *newloc* argument is (**locale_t**)0, the current locale shall not be changed; this value can be used to query the current locale setting. If the *newloc* argument is LC_GLOBAL_LOCALE, any thread-local locale for the calling thread shall be uninstalled; the thread shall again use the global locale as the current locale, and changes to the global locale shall affect the thread. Otherwise, the locale represented by *newloc* shall be installed as a thread-local locale to be used as the current locale for the calling thread.

Once the *uselocale*() function has been called to install a thread-local locale, the behavior of every interface using data from the current locale shall be affected for the calling thread. The current locale for other threads shall remain unchanged.

RETURN VALUE

Upon successful completion, the *uselocale*() function shall return a handle for the thread-local locale that was in use as the current locale for the calling thread on entry to the function, or LC_GLOBAL_LOCALE if no thread-local locale was in use. Otherwise, *uselocale*() shall return (**locale_t**)0 and set *errno* to indicate the error.

ERRORS

The *uselocale*() function may fail if:

EINVAL
newloc is not a valid locale object and is not (**locale_t**)0.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Unlike the *setlocale*() function, the *uselocale*() function does not allow replacing some locale categories only. Applications that need to install a locale which differs only in a few categories must use *newlocale*() to change a locale object equivalent to the currently used locale and install it.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

duplocale( ), *freelocale*( ), *newlocale*( ), *setlocale*( )

The Base Definitions volume of POSIX.1-2017, **<locale.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

utime — set file access and modification times

SYNOPSIS

#include <utime.h>

int utime(const char **path*, const struct utimbuf **times*);

DESCRIPTION

The *utime*() function shall set the access and modification times of the file named by the *path* argument.

If *times* is a null pointer, the access and modification times of the file shall be set to the current time. The effective user ID of the process shall match the owner of the file, or the process has write permission to the file or has appropriate privileges, to use *utime*() in this manner.

If *times* is not a null pointer, *times* shall be interpreted as a pointer to a **utimbuf** structure and the access and modification times shall be set to the values contained in the designated structure. Only a process with the effective user ID equal to the user ID of the file or a process with appropriate privileges may use *utime*() this way.

The **utimbuf** structure is defined in the *<utime.h>* header. The times in the structure **utimbuf** are measured in seconds since the Epoch.

Upon successful completion, the *utime*() function shall mark the last file status change timestamp for update; see *<sys/stat.h>*.

RETURN VALUE

Upon successful completion, 0 shall be returned. Otherwise, −1 shall be returned and *errno* shall be set to indicate the error, and the file times shall not be affected.

ERRORS

The *utime*() function shall fail if:

EACCES
Search permission is denied by a component of the path prefix; or the *times* argument is a null pointer and the effective user ID of the process does not match the owner of the file, the process does not have write permission for the file, and the process does not have appropriate privileges.

ELOOP
A loop exists in symbolic links encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a component of a pathname is longer than {NAME_MAX}.

ENOENT
A component of *path* does not name an existing file or *path* is an empty string.

ENOTDIR
A component of the path prefix names an existing file that is neither a directory nor a symbolic link to a directory, or the *path* argument contains at least one non-<slash> character and ends with one or more trailing <slash> characters and the last pathname component names an existing file that is neither a directory nor a symbolic link to a directory.

EPERM
The *times* argument is not a null pointer and the effective user ID of the calling process does not match the owner of the file and the calling process does not have appropriate privileges.

EROFS
The file system containing the file is read-only.

The *utime*() function may fail if:

ELOOP
More than {SYMLOOP_MAX} symbolic links were encountered during resolution of the *path* argument.

ENAMETOOLONG
The length of a pathname exceeds {PATH_MAX}, or pathname resolution of a symbolic link produced an intermediate result with a length that exceeds {PATH_MAX}.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Since the **utimbuf** structure only contains **time_t** variables and is not accurate to fractions of a second, applications should use the *utimensat*() function instead of the obsolescent *utime*() function.

RATIONALE

The *actime* structure member must be present so that an application may set it, even though an implementation may ignore it and not change the last data access timestamp on the file. If an application intends to leave one of the times of a file unchanged while changing the other, it should use *stat*() or *fstat*() to retrieve the file's *st_atim* and *st_mtim* parameters, set *actime* and *modtime* in the buffer, and change one of them before making the *utime*() call.

FUTURE DIRECTIONS

The *utime*() function may be removed in a future version.

SEE ALSO

fstat( ), *fstatat*( ), *futimens*( )

The Base Definitions volume of POSIX.1-2017, **<sys_stat.h>**, **<utime.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

utimensat, utimes — set file access and modification times

SYNOPSIS

```
#include <sys/stat.h>

int utimensat(int fd, const char *path, const struct timespec times[2],
    int flag);

#include <sys/time.h>

int utimes(const char *path, const struct timeval times[2]);
```

DESCRIPTION

Refer to *futimens*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

va_arg, va_copy, va_end, va_start — handle variable argument list

SYNOPSIS

#include <stdarg.h>

type va_arg(va_list *ap*, *type*);
void va_copy(va_list *dest*, va_list *src*);
void va_end(va_list *ap*);
void va_start(va_list *ap*, *argN*);

DESCRIPTION

Refer to the Base Definitions volume of POSIX.1-2017, **<stdarg.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vdprintf, vfprintf, vprintf, vsnprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vdprintf(int fildes, const char *restrict format, va_list ap);
int vfprintf(FILE *restrict stream, const char *restrict format,
   va_list ap);
int vprintf(const char *restrict format, va_list ap);
int vsnprintf(char *restrict s, size_t n, const char *restrict format,
   va_list ap);
int vsprintf(char *restrict s, const char *restrict format, va_list ap);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *vdprintf*(), *vfprintf*(), *vprintf*(), *vsnprintf*(), and *vsprintf*() functions shall be equivalent to the *dprintf*(), *fprintf*(), *printf*(), *snprintf*(), and *sprintf*() functions respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by *<stdarg.h>*.

These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fprintf*( ).

ERRORS

Refer to *fprintf*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fprintf*( )

The Base Definitions volume of POSIX.1-2017, **<stdarg.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vfscanf, vscanf, vsscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vfscanf(FILE *restrict stream, const char *restrict format,
   va_list arg);
int vscanf(const char *restrict format, va_list arg);
int vsscanf(const char *restrict s, const char *restrict format,
   va_list arg);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *vscanf*(), *vfscanf*(), and *vsscanf*() functions shall be equivalent to the *scanf*(), *fscanf*(), and *sscanf*() functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the *<stdarg.h>* header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fscanf*( ).

ERRORS

Refer to *fscanf*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fscanf*( )

The Base Definitions volume of POSIX.1-2017, **<stdarg.h>**, **<stdio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vfwprintf, vswprintf, vwprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwprintf(FILE *restrict stream, const wchar_t *restrict format,
   va_list arg);
int vswprintf(wchar_t *restrict ws, size_t n,
   const wchar_t *restrict format, va_list arg);
int vwprintf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *vfwprintf*(), *vswprintf*(), and *vwprintf*() functions shall be equivalent to *fwprintf*(), *swprintf*(), and *wprintf*() respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by *<stdarg.h>*.

These functions shall not invoke the *va_end* macro. However, as these functions do invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwprintf*( ).

ERRORS

Refer to *fwprintf*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fwprintf*( )

The Base Definitions volume of POSIX.1-2017, **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vfwscanf, vswscanf, vwscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vfwscanf(FILE *restrict stream, const wchar_t *restrict format,
   va_list arg);
int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
   va_list arg);
int vwscanf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *vfwscanf*(), *vswscanf*(), and *vwscanf*() functions shall be equivalent to the *fwscanf*(), *swscanf*(), and *wscanf*() functions, respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined in the *<stdarg.h>* header. These functions shall not invoke the *va_end* macro. As these functions invoke the *va_arg* macro, the value of *ap* after the return is unspecified.

RETURN VALUE

Refer to *fwscanf*( ).

ERRORS

Refer to *fwscanf*( ).

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Applications using these functions should call *va_end*(*ap*) afterwards to clean up.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.5, *Standard I/O Streams*, *fwscanf*( )

The Base Definitions volume of POSIX.1-2017, **<stdarg.h>**, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vprintf — format the output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vprintf(const char *restrict format, va_list ap);
```

DESCRIPTION

Refer to *vfprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vscanf(const char *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vsnprintf, vsprintf — format output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vsnprintf(char *restrict s, size_t n,
   const char *restrict format, va_list ap);
int vsprintf(char *restrict s, const char *restrict format,
   va_list ap);
```

DESCRIPTION

Refer to *vfprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vsscanf — format input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>

int vsscanf(const char *restrict s, const char *restrict format,
    va_list arg);
```

DESCRIPTION

Refer to *vfscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vswprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vswprintf(wchar_t *restrict ws, size_t n,
   const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfwprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vswscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vswscanf(const wchar_t *restrict ws, const wchar_t *restrict format,
    va_list arg);
```

DESCRIPTION

Refer to *vfwscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vwprintf — wide-character formatted output of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwprintf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfwprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

vwscanf — wide-character formatted input of a stdarg argument list

SYNOPSIS

```
#include <stdarg.h>
#include <stdio.h>
#include <wchar.h>

int vwscanf(const wchar_t *restrict format, va_list arg);
```

DESCRIPTION

Refer to *vfwscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wait, waitpid — wait for a child process to stop or terminate

SYNOPSIS

```
#include <sys/wait.h>

pid_t wait(int *stat_loc);
pid_t waitpid(pid_t pid, int *stat_loc, int options);
```

DESCRIPTION

The *wait*() and *waitpid*() functions shall obtain status information (see *Section 2.13*, *Status Information*) pertaining to one of the caller's child processes. The *wait*() function obtains status information for process termination from any child process. The *waitpid*() function obtains status information for process termination, and optionally process stop and/or continue, from a specified subset of the child processes.

The *wait*() function shall cause the calling thread to become blocked until status information generated by child process termination is made available to the thread, or until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process, or an error occurs. If termination status information is available prior to the call to *wait*(), return shall be immediate. If termination status information is available for two or more child processes, the order in which their status is reported is unspecified.

As described in *Section 2.13*, *Status Information*, the *wait*() and *waitpid*() functions consume the status information they obtain.

The behavior when multiple threads are blocked in *wait*(), *waitid*(), or *waitpid*() is described in *Section 2.13*, *Status Information*.

The *waitpid*() function shall be equivalent to *wait*() if the *pid* argument is (**pid_t**)−1 and the *options* argument is 0. Otherwise, its behavior shall be modified by the values of the *pid* and *options* arguments.

The *pid* argument specifies a set of child processes for which *status* is requested. The *waitpid*() function shall only return the status of a child process from this set:

* If *pid* is equal to (**pid_t**)−1, *status* is requested for any child process. In this respect, *waitpid*() is then equivalent to *wait*().
* If *pid* is greater than 0, it specifies the process ID of a single child process for which *status* is requested.
* If *pid* is 0, *status* is requested for any child process whose process group ID is equal to that of the calling process.
* If *pid* is less than (**pid_t**)−1, *status* is requested for any child process whose process group ID is equal to the absolute value of *pid*.

The *options* argument is constructed from the bitwise-inclusive OR of zero or more of the following flags, defined in the *<sys/wait.h>* header:

WCONTINUED
: The *waitpid*() function shall report the status of any continued child process specified by *pid* whose status has not been reported since it continued from a job control stop.

WNOHANG
: The *waitpid*() function shall not suspend execution of the calling thread if *status* is not immediately available for one of the child processes specified by *pid*.

WUNTRACED
: The status of any child processes specified by *pid* that are stopped, and whose status has not yet been reported since they stopped, shall also be reported to the requesting process.

If *wait*() or *waitpid*() return because the status of a child process is available, these functions shall return a value equal to the process ID of the child process. In this case, if the value of the argument *stat_loc* is not a null pointer, information shall be stored in the location pointed to by *stat_loc*. The value stored at the location pointed to by *stat_loc* shall be 0 if and only if the status returned is from a terminated child process that terminated by one of the following means:

1. The process returned 0 from *main*().
2. The process called *_exit*() or *exit*() with a *status* argument of 0.
3. The process was terminated because the last thread in the process terminated.

Regardless of its value, this information may be interpreted using the following macros, which are defined in <*sys/wait.h*> and evaluate to integral expressions; the *stat_val* argument is the integer value pointed to by *stat_loc*.

WIFEXITED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that terminated normally.

WEXITSTATUS(*stat_val*)
If the value of WIFEXITED(*stat_val*) is non-zero, this macro evaluates to the low-order 8 bits of the *status* argument that the child process passed to *_exit*() or *exit*(), or the value the child process returned from *main*().

WIFSIGNALED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that terminated due to the receipt of a signal that was not caught (see <*signal.h*>).

WTERMSIG(*stat_val*)
If the value of WIFSIGNALED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.

WIFSTOPPED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that is currently stopped.

WSTOPSIG(*stat_val*)
If the value of WIFSTOPPED(*stat_val*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.

WIFCONTINUED(*stat_val*)
Evaluates to a non-zero value if *status* was returned for a child process that has continued from a job control stop.

It is unspecified whether the *status* value returned by calls to *wait*() or *waitpid*() for processes created by *posix_spawn*() or *posix_spawnp*() can indicate a WIFSTOPPED(*stat_val*) before subsequent calls to *wait*() or *waitpid*() indicate WIFEXITED(*stat_val*) as the result of an error detected before the new process image starts executing.

It is unspecified whether the *status* value returned by calls to *wait*() or *waitpid*() for processes created by *posix_spawn*() or *posix_spawnp*() can indicate a WIFSIGNALED(*stat_val*) if a signal is sent to the parent's process group after *posix_spawn*() or *posix_spawnp*() is called.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that specified the WUNTRACED flag and did not specify the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFSTOPPED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that specified the WUNTRACED and WCONTINUED flags, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), WIFSTOPPED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that did not specify the WUNTRACED or WCONTINUED flags, or by a call to the *wait*() function, exactly one of the macros WIFEXITED(**stat_loc*) and WIFSIGNALED(**stat_loc*) shall evaluate to a non-zero value.

If the information pointed to by *stat_loc* was stored by a call to *waitpid*() that did not specify the

WUNTRACED flag and specified the WCONTINUED flag, exactly one of the macros WIFEXITED(**stat_loc*), WIFSIGNALED(**stat_loc*), and WIFCONTINUED(**stat_loc*) shall evaluate to a non-zero value.

If _POSIX_REALTIME_SIGNALS is defined, and the implementation queues the SIGCHLD signal, then if *wait*() or *waitpid*() returns because the status of a child process is available, any pending SIGCHLD signal associated with the process ID of the child process shall be discarded. Any other pending SIGCHLD signals shall remain pending.

Otherwise, if SIGCHLD is blocked, if *wait*() or *waitpid*() return because the status of a child process is available, any pending SIGCHLD signal shall be cleared unless the status of another child process is available.

For all other conditions, it is unspecified whether child *status* will be available when a SIGCHLD signal is delivered.

There may be additional implementation-defined circumstances under which *wait*() or *waitpid*() report *status*. This shall not occur unless the calling process or one of its child processes explicitly makes use of a non-standard extension. In these cases the interpretation of the reported *status* is implementation-defined.

If a parent process terminates without waiting for all of its child processes to terminate, the remaining child processes shall be assigned a new parent process ID corresponding to an implementation-defined system process.

RETURN VALUE

If *wait*() or *waitpid*() returns because the status of a child process is available, these functions shall return a value equal to the process ID of the child process for which *status* is reported. If *wait*() or *waitpid*() returns due to the delivery of a signal to the calling process, −1 shall be returned and *errno* set to **[EINTR]**. If *waitpid*() was invoked with WNOHANG set in *options*, it has at least one child process specified by *pid* for which *status* is not available, and *status* is not available for any process specified by *pid*, 0 is returned. Otherwise, −1 shall be returned, and *errno* set to indicate the error.

ERRORS

The *wait*() function shall fail if:

ECHILD
The calling process has no existing unwaited-for child processes.

EINTR
The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

The *waitpid*() function shall fail if:

ECHILD
The process specified by *pid* does not exist or is not a child of the calling process, or the process group specified by *pid* does not exist or does not have any member process that is a child of the calling process.

EINTR
The function was interrupted by a signal. The value of the location pointed to by *stat_loc* is undefined.

EINVAL
The *options* argument is not valid.

The following sections are informative.

EXAMPLES

Waiting for a Child Process and then Checking its Status

The following example demonstrates the use of *waitpid*(), *fork*(), and the macros used to interpret the status value returned by *waitpid*() (and *wait*()). The code segment creates a child process which does some unspecified work. Meanwhile the parent loops performing calls to *waitpid*() to monitor the status of the child.

The loop terminates when child termination is detected.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
...
pid_t child_pid, wpid;
int status;

child_pid = fork();
if (child_pid == -1) {      /* fork() failed */
    perror("fork");
    exit(EXIT_FAILURE);
}

if (child_pid == 0) {       /* This is the child */
    /* Child does some work and then terminates */
    ...

} else {                    /* This is the parent */
    do {
        wpid = waitpid(child_pid, &status, WUNTRACED
#ifdef WCONTINUED       /* Not all implementations support this */
        | WCONTINUED
#endif
        );
        if (wpid == -1) {
            perror("waitpid");
            exit(EXIT_FAILURE);
        }

        if (WIFEXITED(status)) {
            printf("child exited, status=%d\n", WEXITSTATUS(status));

        } else if (WIFSIGNALED(status)) {
            printf("child killed (signal %d)\n", WTERMSIG(status));

        } else if (WIFSTOPPED(status)) {
            printf("child stopped (signal %d)\n", WSTOPSIG(status));

#ifdef WIFCONTINUED     /* Not all implementations support this */
        } else if (WIFCONTINUED(status)) {
            printf("child continued\n");
#endif
        } else {    /* Non-standard case -- may never happen */
            printf("Unexpected status (0x%x)\n", status);
        }
    } while (!WIFEXITED(status) && !WIFSIGNALED(status));
}
```

Waiting for a Child Process in a Signal Handler for SIGCHLD

The following example demonstrates how to use *waitpid*() in a signal handler for SIGCHLD without passing −1 as the *pid* argument. (See the APPLICATION USAGE section below for the reasons why passing a *pid* of −1 is not recommended.) The method used here relies on the standard behavior of *waitpid*() when SIGCHLD is blocked. On historical non-conforming systems, the status of some child processes might not be reported.

```
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#define CHILDREN 10

static void
handle_sigchld(int signum, siginfo_t *sinfo, void *unused)
{
    int sav_errno = errno;
    int status;

    /*
     * Obtain status information for the child which
     * caused the SIGCHLD signal and write its exit code
     * to stdout.
    */
    if (sinfo->si_code != CLD_EXITED)
    {
        static char msg[] = "wrong si_code\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (waitpid(sinfo->si_pid, &status, 0) = = -1)
    {
        static char msg[] = "waitpid() failed\n";
        write(2, msg, sizeof msg - 1);
    }
    else if (!WIFEXITED(status))
    {
        static char msg[] = "WIFEXITED was false\n";
        write(2, msg, sizeof msg - 1);
    }
    else
    {
        int code = WEXITSTATUS(status);
        char buf[2];
        buf[0] = '0' + code;
        buf[1] = '\n';
        write(1, buf, 2);
    }
    errno = sav_errno;
}

int
main(void)
{
    int i;
    pid_t pid;
    struct sigaction sa;

    sa.sa_flags = SA_SIGINFO;
    sa.sa_sigaction = handle_sigchld;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGCHLD, &sa, NULL) = = -1)
```

```
        {
            perror("sigaction");
            exit(EXIT_FAILURE);
        }

        for (i = 0; i < CHILDREN; i++)
        {
            switch (pid = fork())
            {
            case -1:
                perror("fork");
                exit(EXIT_FAILURE);
            case 0:
                sleep(2);
                _exit(i);
            }
        }

        /* Wait for all the SIGCHLD signals, then terminate on SIGALRM */
        alarm(3);
        for (;;)
            pause();

        return 0; /* NOTREACHED */
    }
```

APPLICATION USAGE

Calls to *wait*() will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system*()). For this and other reasons it is recommended that portable applications not use *wait*(), but instead use *waitpid*(). For these same reasons, the use of *waitpid*() with a *pid* argument of −1, and the use of *waitid*() with the *idtype* argument set to P_ALL, are also not recommended for portable applications.

As specified in *Consequences of Process Termination*, if the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process will not cause status information to become available to a thread blocked in *wait*(), *waitid*(), or *waitpid*(). Thus, a thread blocked in one of the wait functions will remain blocked unless some other condition causes the thread to resume execution (such as an **[ECHILD]** failure due to no remaining children in the set of waited-for children).

RATIONALE

A call to the *wait*() or *waitpid*() function only returns *status* on an immediate child process of the calling process; that is, a child that was produced by a single *fork*() call (perhaps followed by an *exec* or other function calls) from the parent. If a child produces grandchildren by further use of *fork*(), none of those grandchildren nor any of their descendants affect the behavior of a *wait*() from the original parent process. Nothing in this volume of POSIX.1-2017 prevents an implementation from providing extensions that permit a process to get *status* from a grandchild or any other process, but a process that does not use such extensions must be guaranteed to see *status* from only its direct children.

The *waitpid*() function is provided for three reasons:

1. To support job control
2. To permit a non-blocking version of the *wait*() function
3. To permit a library routine, such as *system*() or *pclose*(), to wait for its children without interfering with other terminated children for which the process has not waited

The first two of these facilities are based on the *wait3*( ) function provided by 4.3 BSD. The function uses the *options* argument, which is equivalent to an argument to *wait3*( ). The WUNTRACED flag is used only in conjunction with job control on systems supporting job control. Its name comes from 4.3 BSD and refers to the fact that there are two types of stopped processes in that implementation: processes being traced via the *ptrace*() debugging facility and (untraced) processes stopped by job control signals. Since *ptrace*() is not part of this volume of POSIX.1-2017, only the second type is relevant. The name WUNTRACED was retained because its usage is the same, even though the name is not intuitively meaningful in this context.

The third reason for the *waitpid*() function is to permit independent sections of a process to spawn and wait for children without interfering with each other. For example, the following problem occurs in developing a portable shell, or command interpreter:

```
stream = popen("/bin/true");
(void) system("sleep 100");
(void) pclose(stream);
```

On all historical implementations, the final *pclose*() fails to reap the *wait*() *status* of the *popen*().

The status values are retrieved by macros, rather than given as specific bit encodings as they are in most historical implementations (and thus expected by existing programs). This was necessary to eliminate a limitation on the number of signals an implementation can support that was inherent in the traditional encodings. This volume of POSIX.1-2017 does require that a *status* value of zero corresponds to a process calling *_exit*(0), as this is the most common encoding expected by existing programs. Some of the macro names were adopted from 4.3 BSD.

These macros syntactically operate on an arbitrary integer value. The behavior is undefined unless that value is one stored by a successful call to *wait*() or *waitpid*() in the location pointed to by the *stat_loc* argument. An early proposal attempted to make this clearer by specifying each argument as **stat_loc* rather than *stat_val*. However, that did not follow the conventions of other specifications in this volume of POSIX.1-2017 or traditional usage. It also could have implied that the argument to the macro must literally be **stat_loc*; in fact, that value can be stored or passed as an argument to other functions before being interpreted by these macros.

The extension that affects *wait*() and *waitpid*() and is common in historical implementations is the *ptrace*() function. It is called by a child process and causes that child to stop and return a *status* that appears identical to the *status* indicated by WIFSTOPPED. The *status* of *ptrace*() children is traditionally returned regardless of the WUNTRACED flag (or by the *wait*() function). Most applications do not need to concern themselves with such extensions because they have control over what extensions they or their children use. However, applications, such as command interpreters, that invoke arbitrary processes may see this behavior when those arbitrary processes misuse such extensions.

Implementations that support **core** file creation or other implementation-defined actions on termination of some processes traditionally provide a bit in the *status* returned by *wait*() to indicate that such actions have occurred.

Allowing the *wait*() family of functions to discard a pending SIGCHLD signal that is associated with a successfully waited-for child process puts them into the *sigwait*() and *sigwaitinfo*() category with respect to SIGCHLD.

This definition allows implementations to treat a pending SIGCHLD signal as accepted by the process in *wait*(), with the same meaning of "accepted" as when that word is applied to the *sigwait*() family of functions.

Allowing the *wait*() family of functions to behave this way permits an implementation to be able to deal precisely with SIGCHLD signals.

In particular, an implementation that does accept (discard) the SIGCHLD signal can make the following guarantees regardless of the queuing depth of signals in general (the list of waitable children can hold the SIGCHLD queue):

1. If a SIGCHLD signal handler is established via *sigaction*() without the SA_RESETHAND flag, SIGCHLD signals can be accurately counted; that is, exactly one SIGCHLD signal will be delivered to or accepted by the process for every child process that terminates.

2. A single *wait*() issued from a SIGCHLD signal handler can be guaranteed to return immediately with status information for a child process.
3. When SA_SIGINFO is requested, the SIGCHLD signal handler can be guaranteed to receive a non-null pointer to a **siginfo_t** structure that describes a child process for which a wait via *waitpid*() or *waitid*() will not block or fail.
4. The *system*() function will not cause the SIGCHLD handler of a process to be called as a result of the *fork*()/*exec* executed within *system*() because *system*() will accept the SIGCHLD signal when it performs a *waitpid*() for its child process. This is a desirable behavior of *system*() so that it can be used in a library without causing side-effects to the application linked with the library.

An implementation that does not permit the *wait*() family of functions to accept (discard) a pending SIGCHLD signal associated with a successfully waited-for child, cannot make the guarantees described above for the following reasons:

Guarantee #1

Although it might be assumed that reliable queuing of all SIGCHLD signals generated by the system can make this guarantee, the counter-example is the case of a process that blocks SIGCHLD and performs an indefinite loop of *fork*()/*wait*() operations. If the implementation supports queued signals, then eventually the system will run out of memory for the queue. The guarantee cannot be made because there must be some limit to the depth of queuing.

Guarantees #2 and #3

These cannot be guaranteed unless the *wait*() family of functions accepts the SIGCHLD signal. Otherwise, a *fork*()/*wait*() executed while SIGCHLD is blocked (as in the *system*() function) will result in an invocation of the handler when SIGCHLD is unblocked, after the process has disappeared.

Guarantee #4

Although possible to make this guarantee, *system*() would have to set the SIGCHLD handler to SIG_DFL so that the SIGCHLD signal generated by its *fork*() would be discarded (the SIGCHLD default action is to be ignored), then restore it to its previous setting. This would have the undesirable side-effect of discarding all SIGCHLD signals pending to the process.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.13, *Status Information*, *exec*, *exit*( ), *fork*( ), *system*( ), *waitid*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.12*, *Memory Synchronization*, **<signal.h>**, **<sys_wait.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

waitid — wait for a child process to change state

SYNOPSIS

#include <sys/wait.h>

int waitid(idtype_t *idtype*, id_t *id*, siginfo_t **infop*, int *options*);

DESCRIPTION

The *waitid*() function shall obtain status information (see *Section 2.13*, *Status Information*) pertaining to termination, stop, and/or continue events in one of the caller's child processes.

The *waitid*() function shall cause the calling thread to become blocked until an error occurs or status information becomes available to the calling thread that satisfies all of the following properties (‘‘matching status information’’):

* The status information is from one of the child processes in the set of child processes specified by the *idtype* and *id* arguments.

* The state change in the status information matches one of the state change flags set in the *options* argument.

If matching status information is available prior to the call to *waitid*(), return shall be immediate. If matching status information is available for two or more child processes, the order in which their status is reported is unspecified.

As described in *Section 2.13*, *Status Information*, the *waitid*() function consumes the status information it obtains unless the WNOWAIT flag is set in the *options* argument.

The behavior when multiple threads are blocked in *wait*(), *waitid*(), or *waitpid*() is described in *Section 2.13*, *Status Information*.

The *waitid*() function shall record the obtained status information in the structure pointed to by *infop*. The fields of the structure pointed to by *infop* shall be filled in as described under ‘‘Pointer to a Function’’ in *Section 2.4.3*, *Signal Actions*.

The *idtype* and *id* arguments are used to specify which children *waitid*() waits for.

If *idtype* is P_PID, *waitid*() shall wait for the child with a process ID equal to (**pid_t**)*id*.

If *idtype* is P_PGID, *waitid*() shall wait for any child with a process group ID equal to (**pid_t**)*id*.

If *idtype* is P_ALL, *waitid*() shall wait for any children and *id* is ignored.

The *options* argument is used to specify which state changes *waitid*() shall wait for. It is formed by OR'ing together the following flags:

WCONTINUED	Status shall be returned for any continued child process whose status either has not been reported since it continued from a job control stop or has been reported only by calls to *waitid*() with the WNOWAIT flag set.
WEXITED	Wait for processes that have exited.
WNOHANG	Do not hang if no status is available; return immediately.
WNOWAIT	Keep the process whose status is returned in *infop* in a waitable state. This shall not affect the state of the process; the process may be waited for again after this call completes.
WSTOPPED	Status shall be returned for any child that has stopped upon receipt of a signal, and whose status either has not been reported since it stopped or has been reported only by calls to *waitid*() with the WNOWAIT flag set.

Applications shall specify at least one of the flags WEXITED, WSTOPPED, or WCONTINUED to be OR'ed in with the *options* argument.

The application shall ensure that the *infop* argument points to a **siginfo_t** structure. If *waitid*() returns because a child process was found that satisfied the conditions indicated by the arguments *idtype* and *options*, then the structure pointed to by *infop* shall be filled in by the system with the status of the process; the *si_signo* member shall be set equal to SIGCHLD. If *waitid*() returns because WNOHANG was specified and status is not available for any process specified by *idtype* and *id*, then the *si_signo* and *si_pid* members of the structure pointed to by *infop* shall be set to zero and the values of other members of the structure are unspecified.

RETURN VALUE

If WNOHANG was specified and status is not available for any process specified by *idtype* and *id*, 0 shall be returned. If *waitid*() returns due to the change of state of one of its children, 0 shall be returned. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

The *waitid*() function shall fail if:

ECHILD
The calling process has no existing unwaited-for child processes.

EINTR
The *waitid*() function was interrupted by a signal.

EINVAL
An invalid value was specified for *options*, or *idtype* and *id* specify an invalid set of processes.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

Calls to *waitid*() with *idtype* equal to P_ALL will collect information about any child process. This may result in interactions with other interfaces that may be waiting for their own children (such as by use of *system*()). For this reason it is recommended that portable applications not use *waitid*() with idtype of P_ALL. See also APPLICATION USAGE for *wait*().

As specified in *Consequences of Process Termination*, if the calling process has SA_NOCLDWAIT set or has SIGCHLD set to SIG_IGN, then the termination of a child process will not cause status information to become available to a thread blocked in *wait*(), *waitid*(), or *waitpid*(). Thus, a thread blocked in one of the wait functions will remain blocked unless some other condition causes the thread to resume execution (such as an **[ECHILD]** failure due to no remaining children in the set of waited-for children).

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

Section 2.4.3, *Signal Actions*, *Section 2.13*, *Status Information*, *exec*, *exit*( ), *wait*( )

The Base Definitions volume of POSIX.1-2017, **<signal.h>**, **<sys_wait.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

waitpid — wait for a child process to stop or terminate

SYNOPSIS

#include <sys/wait.h>

pid_t waitpid(pid_t *pid*, int **stat_loc*, int *options*);

DESCRIPTION

Refer to *wait*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcpcpy — copy a wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

Refer to *wcscpy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcpncpy — copy a fixed-size wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

Refer to *wcsncpy*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcrtomb — convert a wide-character code to a character (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t wcrtomb(char *restrict s, wchar_t wc, mbstate_t *restrict ps);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

If *s* is a null pointer, the *wcrtomb*() function shall be equivalent to the call:

```
wcrtomb(buf, L'\0', ps)
```

where *buf* is an internal buffer.

If *s* is not a null pointer, the *wcrtomb*() function shall determine the number of bytes needed to represent the character that corresponds to the wide character given by *wc* (including any shift sequences), and store the resulting bytes in the array whose first element is pointed to by *s*. At most {MB_CUR_MAX} bytes are stored. If *wc* is a null wide character, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state. The resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcrtomb*() function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence. The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *wcrtomb*().

The *wcrtomb*() function need not be thread-safe if called with a NULL *ps* argument.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

The *wcrtomb*() function shall not change the setting of *errno* if successful.

RETURN VALUE

The *wcrtomb*() function shall return the number of bytes stored in the array object (including any shift sequences). When *wc* is not a valid wide character, an encoding error shall occur. In this case, the function shall store the value of the macro **[EILSEQ]** in *errno* and shall return (**size_t**)−1; the conversion state shall be undefined.

ERRORS

The *wcrtomb*() function shall fail if:

EILSEQ
An invalid wide-character code is detected.

The *wcrtomb*() function may fail if:

EINVAL
ps points to an object that contains an invalid conversion state.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *wcsrtombs*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcscasecmp, wcscasecmp_l, wcsncasecmp, wcsncasecmp_l — case-insensitive wide-character string comparison

SYNOPSIS

```
#include <wchar.h>

int wcscasecmp(const wchar_t *ws1, const wchar_t *ws2);
int wcscasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
   locale_t locale);
int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
   size_t n, locale_t locale);
```

DESCRIPTION

The *wcscasecmp*() and *wcsncasecmp*() functions are the wide-character equivalent of the *strcasecmp*() and *strncasecmp*() functions, respectively.

The *wcscasecmp*() and *wcscasecmp_l*() functions shall compare, while ignoring differences in case, the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcsncasecmp*() and *wcsncasecmp_l*() functions shall compare, while ignoring differences in case, not more than *n* wide-characters from the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The *wcscasecmp*() and *wcsncasecmp*() functions use the current locale to determine the case of the wide characters.

The *wcscasecmp_l*() and *wcsncasecmp_l*() functions use the locale represented by *locale* to determine the case of the wide characters.

When the *LC_CTYPE* category of the locale being used is from the POSIX locale, these functions shall behave as if the wide-character strings had been converted to lowercase and then a comparison of wide-character codes performed. Otherwise, the results are unspecified.

The information for *wcscasecmp_l*() and *wcsncasecmp_l*() about the case of the characters comes from the locale represented by *locale*.

The behavior is undefined if the *locale* argument to *wcscasecmp_l*() or *wcsncasecmp_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon completion, the *wcscasecmp*() and *wcscasecmp_l*() functions shall return an integer greater than, equal to, or less than 0 if the wide-character string pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

Upon completion, the *wcsncasecmp*() and *wcsncasecmp_l*() functions shall return an integer greater than, equal to, or less than 0 if the possibly null wide-character terminated string pointed to by *ws1* is, ignoring case, greater than, equal to, or less than the possibly null wide-character terminated string pointed to by *ws2*, respectively.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcasecmp( ), *wcscmp*( ), *wcsncmp*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcscat — concatenate two wide-character strings

SYNOPSIS

#include <wchar.h>

wchar_t *wcscat(wchar_t *restrict *ws1*, const wchar_t *restrict *ws2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcscat*() function shall append a copy of the wide-character string pointed to by *ws2* (including the terminating null wide-character code) to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcscat*() function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsncat( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcschr — wide-character string scanning operation

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcschr(const wchar_t *ws, wchar_t wc);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcschr*() function shall locate the first occurrence of *wc* in the wide-character string pointed to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code is considered to be part of the wide-character string.

RETURN VALUE

Upon completion, *wcschr*() shall return a pointer to the wide-character code, or a null pointer if the wide-character code is not found.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsrchr( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcscmp — compare two wide-character strings

SYNOPSIS

```
#include <wchar.h>

int wcscmp(const wchar_t *ws1, const wchar_t *ws2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcscmp*() function shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE

Upon completion, *wcscmp*() shall return an integer greater than, equal to, or less than 0, if the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscasecmp( ), *wcsncmp*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcscoll, wcscoll_l — wide-character string comparison using collating information

SYNOPSIS

```
#include <wchar.h>

int wcscoll(const wchar_t *ws1, const wchar_t *ws2);
int wcscoll_l(const wchar_t *ws1, const wchar_t *ws2,
    locale_t locale);
```

DESCRIPTION

For *wcscoll*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcscoll*() and *wcscoll_l*() functions shall compare the wide-character string pointed to by *ws1* to the wide-character string pointed to by *ws2*, both interpreted as appropriate to the *LC_COLLATE* category of the current locale, or the locale represented by *locale*, respectively.

The *wcscoll*() and *wcscoll_l*() functions shall not change the setting of *errno* if successful.

An application wishing to check for error situations should set *errno* to 0 before calling *wcscoll*() or *wcscoll_l*(). If *errno* is non-zero on return, an error has occurred.

The behavior is undefined if the *locale* argument to *wcscoll_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

Upon successful completion, *wcscoll*() and *wcscoll_l*() shall return an integer greater than, equal to, or less than 0, according to whether the wide-character string pointed to by *ws1* is greater than, equal to, or less than the wide-character string pointed to by *ws2*, when both are interpreted as appropriate to the current locale, or to the locale represented by *locale*, respectively. On error, *wcscoll*() and *wcscoll_l*() shall set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

EINVAL
: The *ws1* or *ws2* arguments contain wide-character codes outside the domain of the collating sequence.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *wcsxfrm*() and *wcscmp*() functions should be used for sorting large lists.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcsxfrm*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcpcpy, wcscpy — copy a wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcpcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
wchar_t *wcscpy(wchar_t *restrict ws1, const wchar_t *restrict ws2);
```

DESCRIPTION

For *wcscpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcpcpy*() and *wcscpy*() functions shall copy the wide-character string pointed to by *ws2* (including the terminating null wide-character code) into the array pointed to by *ws1*.

The application shall ensure that there is room for at least *wcslen*(*ws2*)+1 wide characters in the *ws1* array, and that the *ws2* and *ws1* arrays do not overlap.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcpcpy*() function shall return a pointer to the terminating null wide-character code copied into the *ws1* buffer.

The *wcscpy*() function shall return *ws1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strcpy( ), *wcsdup*( ), *wcsncpy*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced

during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcscspn — get the length of a complementary wide substring

SYNOPSIS

#include <wchar.h>

size_t wcscspn(const wchar_t **ws1*, const wchar_t **ws2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcscspn*() function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes *not* from the wide-character string pointed to by *ws2*.

RETURN VALUE

The *wcscspn*() function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcsspn( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsdup — duplicate a wide-character string

SYNOPSIS

#include <wchar.h>

wchar_t *wcsdup(const wchar_t **string*);

DESCRIPTION

The *wcsdup*() function is the wide-character equivalent of the *strdup*() function.

The *wcsdup*() function shall return a pointer to a new wide-character string, allocated as if by a call to *malloc*(), which is the duplicate of the wide-character string *string*. The returned pointer can be passed to *free*(). A null pointer is returned if the new wide-character string cannot be created.

RETURN VALUE

Upon successful completion, the *wcsdup*() function shall return a pointer to the newly allocated wide-character string. Otherwise, it shall return a null pointer and set *errno* to indicate the error.

ERRORS

The *wcsdup*() function shall fail if:

ENOMEM
Memory large enough for the duplicate string could not be allocated.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

For functions that allocate memory as if by *malloc*(), the application should release such memory when it is no longer required by a call to *free*(). For *wcsdup*(), this is the return value.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

free( ), *strdup*( ), *wcscpy*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsftime — convert date and time to a wide-character string

SYNOPSIS

```
#include <wchar.h>

size_t wcsftime(wchar_t *restrict wcs, size_t maxsize,
   const wchar_t *restrict format, const struct tm *restrict timeptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsftime*() function shall be equivalent to the *strftime*() function, except that:

* The argument *wcs* points to the initial element of an array of wide characters into which the generated output is to be placed.
* The argument *maxsize* indicates the maximum number of wide characters to be placed in the output array.
* The argument *format* is a wide-character string and the conversion specifications are replaced by corresponding sequences of wide characters. It is unspecified whether an encoding error occurs if the format string contains **wchar_t** values that do not correspond to members of the character set of the current locale.
* Field widths specify the number of wide characters instead of the number of bytes.
* The return value indicates the number of wide characters placed in the output array.

If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

If the total number of resulting wide-character codes including the terminating null wide-character code is no more than *maxsize*, *wcsftime*() shall return the number of wide-character codes placed into the array pointed to by *wcs*, not including the terminating null wide-character code. Otherwise, zero is returned and the contents of the array are unspecified.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strftime( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcslen, wcsnlen — get length of a fixed-sized wide-character string

SYNOPSIS

```
#include <wchar.h>

size_t wcslen(const wchar_t *ws);
size_t wcsnlen(const wchar_t *ws, size_t maxlen);
```

DESCRIPTION

For *wcslen*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcslen*() function shall compute the number of wide-character codes in the wide-character string to which *ws* points, not including the terminating null wide-character code.

The *wcsnlen*() function shall compute the smaller of the number of wide characters in the array to which *ws* points, not including any terminating null wide-character code, and the value of *maxlen*. The *wcsnlen*() function shall never examine more than the first *maxlen* characters of the wide-character array pointed to by *ws*.

RETURN VALUE

The *wcslen*() function shall return the length of *ws*.

The *wcsnlen*() function shall return the number of wide characters preceding the first null wide-character code in the array to which *ws* points, if *ws* contains a null wide-character code within the first *maxlen* wide characters; otherwise, it shall return *maxlen*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strlen( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsncasecmp, wcsncasecmp_l — case-insensitive wide-character string comparison

SYNOPSIS

```
#include <wchar.h>

int wcsncasecmp(const wchar_t *ws1, const wchar_t *ws2, size_t n);
int wcsncasecmp_l(const wchar_t *ws1, const wchar_t *ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

Refer to *wcscasecmp*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsncat — concatenate a wide-character string with part of another

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsncat(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsncat*() function shall append not more than *n* wide-character codes (a null wide-character code and wide-character codes that follow it are not appended) from the array pointed to by *ws2* to the end of the wide-character string pointed to by *ws1*. The initial wide-character code of *ws2* shall overwrite the null wide-character code at the end of *ws1*. A terminating null wide-character code shall always be appended to the result. If copying takes place between objects that overlap, the behavior is undefined.

RETURN VALUE

The *wcsncat*() function shall return *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscat( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsncmp — compare part of two wide-character strings

SYNOPSIS

#include <wchar.h>

int wcsncmp(const wchar_t **ws1*, const wchar_t **ws2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsncmp*() function shall compare not more than *n* wide-character codes (wide-character codes that follow a null wide-character code are not compared) from the array pointed to by *ws1* to the array pointed to by *ws2*.

The sign of a non-zero return value shall be determined by the sign of the difference between the values of the first pair of wide-character codes that differ in the objects being compared.

RETURN VALUE

Upon successful completion, *wcsncmp*() shall return an integer greater than, equal to, or less than 0, if the possibly null-terminated array pointed to by *ws1* is greater than, equal to, or less than the possibly null-terminated array pointed to by *ws2*, respectively.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscasecmp( ), *wcscmp*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcpncpy, wcsncpy — copy a fixed-size wide-character string, returning a pointer to its end

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcpncpy(wchar_t restrict *ws1, const wchar_t *restrict ws2,
   size_t n);
wchar_t *wcsncpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
   size_t n);
```

DESCRIPTION

For *wcsncpy*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcpncpy*() and *wcsncpy*() functions shall copy not more than *n* wide-character codes (wide-character codes that follow a null wide-character code are not copied) from the array pointed to by *ws2* to the array pointed to by *ws1*. If copying takes place between objects that overlap, the behavior is undefined.

If the array pointed to by *ws2* is a wide-character string that is shorter than *n* wide-character codes, null wide-character codes shall be appended to the copy in the array pointed to by *ws1*, until *n* wide-character codes in all are written.

RETURN VALUE

If any null wide-character codes were written into the destination, the *wcpncpy*() function shall return the address of the first such null wide-character code. Otherwise, it shall return *&ws1*[*n*].

The *wcsncpy*() function shall return *ws1*.

No return values are reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If there is no null wide-character code in the first *n* wide-character codes of the array pointed to by *ws2*, the result is not null-terminated.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

strncpy( ), *wcscpy*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and

The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsnlen — get length of a fixed-sized wide-character string

SYNOPSIS

#include <wchar.h>

size_t wcsnlen(const wchar_t **ws*, size_t *maxlen*);

DESCRIPTION

Refer to *wcslen*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsnrtombs — convert wide-character string to multi-byte string

SYNOPSIS

#include <wchar.h>

size_t wcsnrtombs(char *restrict *dst*, const wchar_t **restrict *src*,
  size_t *nwc*, size_t *len*, mbstate_t *restrict *ps*);

DESCRIPTION

Refer to *wcsrtombs*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcspbrk — scan a wide-character string for a wide-character code

SYNOPSIS

#include <wchar.h>

wchar_t *wcspbrk(const wchar_t **ws1*, const wchar_t **ws2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcspbrk*() function shall locate the first occurrence in the wide-character string pointed to by *ws1* of any wide-character code from the wide-character string pointed to by *ws2*.

RETURN VALUE

Upon successful completion, *wcspbrk*() shall return a pointer to the wide-character code or a null pointer if no wide-character code from *ws2* occurs in *ws1*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( ), *wcsrchr*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsrchr — wide-character string scanning operation

SYNOPSIS

#include <wchar.h>

wchar_t *wcsrchr(const wchar_t **ws*, wchar_t *wc*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsrchr*() function shall locate the last occurrence of *wc* in the wide-character string pointed to by *ws*. The application shall ensure that the value of *wc* is a character representable as a type **wchar_t** and a wide-character code corresponding to a valid character in the current locale. The terminating null wide-character code shall be considered to be part of the wide-character string.

RETURN VALUE

Upon successful completion, *wcsrchr*() shall return a pointer to the wide-character code or a null pointer if *wc* does not occur in the wide-character string.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsnrtombs, wcsrtombs — convert a wide-character string to a character string (restartable)

SYNOPSIS

```
#include <wchar.h>

size_t wcsnrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t nwc, size_t len, mbstate_t *restrict ps);
size_t wcsrtombs(char *restrict dst, const wchar_t **restrict src,
    size_t len, mbstate_t *restrict ps);
```

DESCRIPTION

For *wcsrtombs*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsrtombs*() function shall convert a sequence of wide characters from the array indirectly pointed to by *src* into a sequence of corresponding characters, beginning in the conversion state described by the object pointed to by *ps*. If *dst* is not a null pointer, the converted characters shall then be stored into the array pointed to by *dst*. Conversion continues up to and including a terminating null wide character, which shall also be stored. Conversion shall stop earlier in the following cases:

* When a code is reached that does not correspond to a valid character
* When the next character would exceed the limit of *len* total bytes to be stored in the array pointed to by *dst* (and *dst* is not a null pointer)

Each conversion shall take place as if by a call to the *wcrtomb*() function.

If *dst* is not a null pointer, the pointer object pointed to by *src* shall be assigned either a null pointer (if conversion stopped due to reaching a terminating null wide character) or the address just past the last wide character converted (if any). If conversion stopped due to reaching a terminating null wide character, the resulting state described shall be the initial conversion state.

If *ps* is a null pointer, the *wcsrtombs*() function shall use its own internal **mbstate_t** object, which is initialized at program start-up to the initial conversion state. Otherwise, the **mbstate_t** object pointed to by *ps* shall be used to completely describe the current conversion state of the associated character sequence.

The *wcsnrtombs*() and *wcsrtombs*() functions need not be thread-safe if called with a NULL *ps* argument.

The *wcsnrtombs*() function shall be equivalent to the *wcsrtombs*() function, except that the conversion is limited to the first *nwc* wide characters.

The *wcsrtombs*() function shall not change the setting of *errno* if successful.

The behavior of these functions shall be affected by the *LC_CTYPE* category of the current locale.

The implementation shall behave as if no function defined in System Interfaces volume of POSIX.1-2017 calls these functions.

RETURN VALUE

If conversion stops because a code is reached that does not correspond to a valid character, an encoding error occurs. In this case, these functions shall store the value of the macro **[EILSEQ]** in *errno* and return (**size_t**)−1; the conversion state is undefined. Otherwise, these functions shall return the number of bytes in the resulting character sequence, not including the terminating null (if any).

ERRORS

These functions shall fail if:

EILSEQ
A wide-character code does not correspond to a valid character.

These functions may fail if:

EINVAL
ps points to an object that contains an invalid conversion state.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mbsinit( ), *wcrtomb*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsspn — get the length of a wide substring

SYNOPSIS

#include <wchar.h>

size_t wcsspn(const wchar_t ***ws1*, const wchar_t ***ws2*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsspn*() function shall compute the length (in wide characters) of the maximum initial segment of the wide-character string pointed to by *ws1* which consists entirely of wide-character codes from the wide-character string pointed to by *ws2*.

RETURN VALUE

The *wcsspn*() function shall return the length of the initial substring of *ws1*; no return value is reserved to indicate an error.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscspn( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsstr — find a wide-character substring

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcsstr(const wchar_t *restrict ws1,
   const wchar_t *restrict ws2);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsstr*() function shall locate the first occurrence in the wide-character string pointed to by *ws1* of the sequence of wide characters (excluding the terminating null wide character) in the wide-character string pointed to by *ws2*.

RETURN VALUE

Upon successful completion, *wcsstr*() shall return a pointer to the located wide-character string, or a null pointer if the wide-character string is not found.

If *ws2* points to a wide-character string with zero length, the function shall return *ws1*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcschr( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstod, wcstof, wcstold — convert a wide-character string to a double-precision number

SYNOPSIS

```
#include <wchar.h>

double wcstod(const wchar_t *restrict nptr, wchar_t **restrict endptr);
float wcstof(const wchar_t *restrict nptr, wchar_t **restrict endptr);
long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **double**, **float**, and **long double** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*())
2. A subject sequence interpreted as a floating-point constant or representing infinity or NaN
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to a floating-point number, and return the result.

The expected form of the subject sequence is an optional **'+'** or **'−'** sign, then one of the following:

* A non-empty sequence of decimal digits optionally containing a radix character; then an optional exponent part consisting of the wide character **'e'** or the wide character **'E'**, optionally followed by a **'+'** or **'−'** wide character, and then followed by one or more decimal digits
* A 0x or 0X, then a non-empty sequence of hexadecimal digits optionally containing a radix character; then an optional binary exponent part consisting of the wide character **'p'** or the wide character **'P'**, optionally followed by a **'+'** or **'−'** wide character, and then followed by one or more decimal digits
* One of INF or INFINITY, or any other wide string equivalent except for case
* One of NAN or NAN(*n-wchar-sequence*$_{opt}$), or any other wide string ignoring case in the NAN part, where:

```
n-wchar-sequence:
    digit
    nondigit
    n-wchar-sequence digit
    n-wchar-sequence nondigit
```

The subject sequence is defined as the longest initial subsequence of the input wide string, starting with the first non-white-space wide character, that is of the expected form. The subject sequence contains no wide characters if the input wide string is not of the expected form.

If the subject sequence has the expected form for a floating-point number, the sequence of wide characters starting with the first digit or the radix character (whichever occurs first) shall be interpreted as a floating constant according to the rules of the C language, except that the radix character shall be used in place of a period, and that if neither an exponent part nor a radix character appears in a decimal floating-point number, or if a binary exponent part does not appear in a hexadecimal floating-point number, an exponent part of the appropriate type with value zero shall be assumed to follow the last digit in the string. If the subject sequence begins with a <hyphen-minus>, the sequence shall be interpreted as negated. A wide-character sequence INF or INFINITY shall be interpreted as an infinity, if representable in the return type, else as if it were a floating constant that is too large for the range of the return type. A wide-character sequence NAN or NAN(*n-wchar-sequence*$_{opt}$) shall be interpreted as a quiet NaN, if supported in the return type, else as if it were a subject sequence part that does not have the expected form; the meaning of the *n*-wchar sequences is implementation-defined. A pointer to the final wide string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

If the subject sequence has the hexadecimal form and FLT_RADIX is a power of 2, the conversion shall be rounded in an implementation-defined manner.

The radix character shall be as defined in the current locale (category *LC_NUMERIC*). In the POSIX locale, or in a locale where the radix character is not defined, the radix character shall default to a <period> ('**.**').

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0 is returned on error and is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstod*(), *wcstof*(), or *wcstold*(), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value. If no conversion could be performed, 0 shall be returned and *errno* may be set to **[EINVAL]**.

If the correct value is outside the range of representable values, ±HUGE_VAL, ±HUGE_VALF, or ±HUGE_VALL shall be returned (according to the sign of the value), and *errno* shall be set to **[ERANGE]**.

If the correct value would cause underflow, a value whose magnitude is no greater than the smallest normalized positive number in the return type shall be returned and *errno* set to **[ERANGE]**.

ERRORS

The *wcstod*() function shall fail if:

ERANGE
The value to be returned would cause overflow or underflow.

The *wcstod*() function may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

If the subject sequence has the hexadecimal form and FLT_RADIX is not a power of 2, and the result is not exactly representable, the result should be one of the two numbers in the appropriate internal format that are adjacent to the hexadecimal floating source value, with the extra stipulation that the error should have a correct sign for the current rounding direction.

If the subject sequence has the decimal form and at most DECIMAL_DIG (defined in <*float.h*>) significant digits, the result should be correctly rounded. If the subject sequence *D* has the decimal form and more than DECIMAL_DIG significant digits, consider the two bounding, adjacent decimal strings *L* and *U*, both having DECIMAL_DIG significant digits, such that the values of *L*, *D*, and *U* satisfy **"L<=D<=U"**. The result should be one of the (equal or adjacent) values that would be obtained by correctly rounding *L* and *U* according to the current rounding direction, with the extra stipulation that the error with respect to *D* should have a correct sign for the current rounding direction.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *iswspace*( ), *localeconv*( ), *setlocale*( ), *wcstol*( )

The Base Definitions volume of POSIX.1-2017, *Chapter 7*, *Locale*, **<float.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstoimax, wcstoumax — convert a wide-character string to an integer type

SYNOPSIS

```
#include <stddef.h>
#include <inttypes.h>

intmax_t wcstoimax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
uintmax_t wcstoumax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall be equivalent to the *wcstol*(), *wcstoll*(), *wcstoul*(), and *wcstoull*() functions, respectively, except that the initial portion of the wide string shall be converted to **intmax_t** and **uintmax_t** representation, respectively.

RETURN VALUE

These functions shall return the converted value, if any.

If no conversion could be performed, zero shall be returned. If the correct value is outside the range of representable values, {INTMAX_MAX}, {INTMAX_MIN}, or {UINTMAX_MAX} shall be returned (according to the return type and sign of the value, if any), and *errno* shall be set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcstol( ), *wcstoul*( )

The Base Definitions volume of POSIX.1-2017, **<inttypes.h>**, **<stddef.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstok — split a wide-character string into tokens

SYNOPSIS

```
#include <wchar.h>

wchar_t *wcstok(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    wchar_t **restrict ptr);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

A sequence of calls to *wcstok*() shall break the wide-character string pointed to by *ws1* into a sequence of tokens, each of which shall be delimited by a wide-character code from the wide-character string pointed to by *ws2*. The *ptr* argument points to a caller-provided **wchar_t** pointer into which the *wcstok*() function shall store information necessary for it to continue scanning the same wide-character string.

The first call in the sequence has *ws1* as its first argument, and is followed by calls with a null pointer as their first argument. The separator string pointed to by *ws2* may be different from call to call.

The first call in the sequence shall search the wide-character string pointed to by *ws1* for the first wide-character code that is *not* contained in the current separator string pointed to by *ws2*. If no such wide-character code is found, then there are no tokens in the wide-character string pointed to by *ws1* and *wcstok*() shall return a null pointer. If such a wide-character code is found, it shall be the start of the first token.

The *wcstok*() function shall then search from there for a wide-character code that *is* contained in the current separator string. If no such wide-character code is found, the current token extends to the end of the wide-character string pointed to by *ws1*, and subsequent searches for a token shall return a null pointer. If such a wide-character code is found, it shall be overwritten by a null wide character, which terminates the current token. The *wcstok*() function shall save a pointer to the following wide-character code, from which the next search for a token shall start.

Each subsequent call, with a null pointer as the value of the first argument, shall start searching from the saved pointer and behave as described above.

The implementation shall behave as if no function calls *wcstok*().

RETURN VALUE

Upon successful completion, the *wcstok*() function shall return a pointer to the first wide-character code of a token. Otherwise, if there is no token, *wcstok*() shall return a null pointer.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstol, wcstoll — convert a wide-character string to a long integer

SYNOPSIS

```
#include <wchar.h>

long wcstol(const wchar_t *restrict nptr, wchar_t **restrict endptr,
   int base);
long long wcstoll(const wchar_t *restrict nptr,
   wchar_t **restrict endptr, int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

These functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **long** and **long long**, respectively. First, they shall decompose the input string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*())
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a **'+'** or **'−'** sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix **'0'** optionally followed by a sequence of the digits **'0'** to **'7'** only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters **'a'** (or **'A'**) to **'f'** (or **'F'**) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a **'+'** or **'−'** sign, but not including an integer suffix. The letters from **'a'** (or **'A'**) to **'z'** (or **'Z'**) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character code representations of 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first non-white-space wide-character code that is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character code, or if the first non-white-space wide-character code is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0, {LONG_MIN} or {LLONG_MIN} and {LONG_MAX} or {LLONG_MAX} are returned on error and are also valid returns on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstol*() or *wcstoll*(), then check *errno*.

RETURN VALUE

Upon successful completion, these functions shall return the converted value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If the correct value is outside the range of representable values, {LONG_MIN}, {LONG_MAX}, {LLONG_MIN}, or {LLONG_MAX} shall be returned (according to the sign of the value), and *errno* set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *iswalpha*( ), *wcstod*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstold — convert a wide-character string to a double-precision number

SYNOPSIS

```
#include <wchar.h>

long double wcstold(const wchar_t *restrict nptr,
    wchar_t **restrict endptr);
```

DESCRIPTION

Refer to *wcstod*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstoll — convert a wide-character string to a long integer

SYNOPSIS

```
#include <wchar.h>

long long wcstoll(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

Refer to *wcstol*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstombs — convert a wide-character string to a character string

SYNOPSIS

```
#include <stdlib.h>

size_t wcstombs(char *restrict s, const wchar_t *restrict pwcs,
    size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcstombs*() function shall convert the sequence of wide-character codes that are in the array pointed to by *pwcs* into a sequence of characters that begins in the initial shift state and store these characters into the array pointed to by *s*, stopping if a character would exceed the limit of *n* total bytes or if a null byte is stored. Each wide-character code shall be converted as if by a call to *wctomb*(), except that the shift state of *wctomb*() shall not be affected.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

No more than *n* bytes shall be modified in the array pointed to by *s*. If copying takes place between objects that overlap, the behavior is undefined. If *s* is a null pointer, *wcstombs*() shall return the length required to convert the entire array regardless of the value of *n*, but no values are stored.

RETURN VALUE

If a wide-character code is encountered that does not correspond to a valid character (of one or more bytes each), *wcstombs*() shall return (**size_t**)−1. Otherwise, *wcstombs*() shall return the number of bytes stored in the character array, not including any terminating null byte. The array shall not be null-terminated if the value returned is *n*.

ERRORS

The *wcstombs*() function shall fail if:

EILSEQ

A wide-character code does not correspond to a valid character.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *mbstowcs*( ), *wctomb*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base

Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstoul, wcstoull — convert a wide-character string to an unsigned long

SYNOPSIS

```
#include <wchar.h>

unsigned long wcstoul(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
unsigned long long wcstoull(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcstoul*() and *wcstoull*() functions shall convert the initial portion of the wide-character string pointed to by *nptr* to **unsigned long** and **unsigned long long** representation, respectively. First, they shall decompose the input wide-character string into three parts:

1. An initial, possibly empty, sequence of white-space wide-character codes (as specified by *iswspace*())
2. A subject sequence interpreted as an integer represented in some radix determined by the value of *base*
3. A final wide-character string of one or more unrecognized wide-character codes, including the terminating null wide-character code of the input wide-character string

Then they shall attempt to convert the subject sequence to an unsigned integer, and return the result.

If *base* is 0, the expected form of the subject sequence is that of a decimal constant, octal constant, or hexadecimal constant, any of which may be preceded by a **'+'** or **'–'** sign. A decimal constant begins with a non-zero digit, and consists of a sequence of decimal digits. An octal constant consists of the prefix **'0'** optionally followed by a sequence of the digits **'0'** to **'7'** only. A hexadecimal constant consists of the prefix 0x or 0X followed by a sequence of the decimal digits and letters **'a'** (or **'A'**) to **'f'** (or **'F'**) with values 10 to 15 respectively.

If the value of *base* is between 2 and 36, the expected form of the subject sequence is a sequence of letters and digits representing an integer with the radix specified by *base*, optionally preceded by a **'+'** or **'–'** sign, but not including an integer suffix. The letters from **'a'** (or **'A'**) to **'z'** (or **'Z'**) inclusive are ascribed the values 10 to 35; only letters whose ascribed values are less than that of *base* shall be permitted. If the value of *base* is 16, the wide-character codes 0x or 0X may optionally precede the sequence of letters and digits, following the sign if present.

The subject sequence is defined as the longest initial subsequence of the input wide-character string, starting with the first wide-character code that is not white space and is of the expected form. The subject sequence contains no wide-character codes if the input wide-character string is empty or consists entirely of white-space wide-character codes, or if the first wide-character code that is not white space is other than a sign or a permissible letter or digit.

If the subject sequence has the expected form and *base* is 0, the sequence of wide-character codes starting with the first digit shall be interpreted as an integer constant. If the subject sequence has the expected form and the value of *base* is between 2 and 36, it shall be used as the base for conversion, ascribing to each letter its value as given above. If the subject sequence begins with a <hyphen-minus>, the value resulting from the conversion shall be negated. A pointer to the final wide-character string shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

In other than the C or POSIX locale, additional locale-specific subject sequence forms may be accepted.

If the subject sequence is empty or does not have the expected form, no conversion shall be performed; the value of *nptr* shall be stored in the object pointed to by *endptr*, provided that *endptr* is not a null pointer.

These functions shall not change the setting of *errno* if successful.

Since 0, {ULONG_MAX}, and {ULLONG_MAX} are returned on error and 0 is also a valid return on success, an application wishing to check for error situations should set *errno* to 0, then call *wcstoul*() or *wcstoull*(), then check *errno*.

RETURN VALUE

Upon successful completion, the *wcstoul*() and *wcstoull*() functions shall return the converted value, if any. If no conversion could be performed, 0 shall be returned and *errno* may be set to indicate the error. If the correct value is outside the range of representable values, {ULONG_MAX} or {ULLONG_MAX} respectively shall be returned and *errno* set to **[ERANGE]**.

ERRORS

These functions shall fail if:

EINVAL
The value of *base* is not supported.

ERANGE
The value to be returned is not representable.

These functions may fail if:

EINVAL
No conversion could be performed.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

fscanf( ), *iswalpha*( ), *wcstod*( ), *wcstol*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcstoumax — convert a wide-character string to an integer type

SYNOPSIS

```
#include <stddef.h>
#include <inttypes.h>

uintmax_t wcstoumax(const wchar_t *restrict nptr,
    wchar_t **restrict endptr, int base);
```

DESCRIPTION

Refer to *wcstoimax*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcswidth — number of column positions of a wide-character string

SYNOPSIS

#include <wchar.h>

int wcswidth(const wchar_t **pwcs*, size_t *n*);

DESCRIPTION

The *wcswidth*() function shall determine the number of column positions required for *n* wide-character codes (or fewer than *n* wide-character codes if a null wide-character code is encountered before *n* wide-character codes are exhausted) in the string pointed to by *pwcs*.

RETURN VALUE

The *wcswidth*() function either shall return 0 (if *pwcs* points to a null wide-character code), or return the number of column positions to be occupied by the wide-character string pointed to by *pwcs*, or return −1 (if any of the first *n* wide-character codes in the wide-character string pointed to by *pwcs* is not a printable wide-character code).

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899: 1990/Amendment 1: 1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcwidth( )

The Base Definitions volume of POSIX.1-2017, *Section 3.103*, *Column Position*, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcsxfrm, wcsxfrm_l — wide-character string transformation

SYNOPSIS

```
#include <wchar.h>

size_t wcsxfrm(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n);
size_t wcsxfrm_l(wchar_t *restrict ws1, const wchar_t *restrict ws2,
    size_t n, locale_t locale);
```

DESCRIPTION

For *wcsxfrm*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wcsxfrm*() and *wcsxfrm_l*() functions shall transform the wide-character string pointed to by *ws2* and place the resulting wide-character string into the array pointed to by *ws1*. The transformation shall be such that if *wcscmp*() is applied to two transformed wide strings, it shall return a value greater than, equal to, or less than 0, corresponding to the result of *wcscoll*() and *wcscoll_l*() applied to the same two original wide-character strings, and the same *LC_COLLATE* category of the current locale or the locale object *locale*, respectively. No more than *n* wide-character codes shall be placed into the resulting array pointed to by *ws1*, including the terminating null wide-character code. If *n* is 0, *ws1* is permitted to be a null pointer. If copying takes place between objects that overlap, the behavior is undefined.

The *wcsxfrm*() and *wcsxfrm_l*() functions shall not change the setting of *errno* if successful.

Since no return value is reserved to indicate an error, an application wishing to check for error situations should set *errno* to 0, then call *wcsxfrm*() or *wcsxfrm_l*(), then check *errno*.

The behavior is undefined if the *locale* argument to *wcsxfrm_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *wcsxfrm*() and *wcsxfrm_l*() functions shall return the length of the transformed wide-character string (not including the terminating null wide-character code). If the value returned is *n* or more, the contents of the array pointed to by *ws1* are unspecified.

On error, the *wcsxfrm*() and *wcsxfrm_l*() functions may set *errno*, but no return value is reserved to indicate an error.

ERRORS

These functions may fail if:

EINVAL

The wide-character string pointed to by *ws2* contains wide-character codes outside the domain of the collating sequence.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The transformation function is such that two transformed wide-character strings can be ordered by *wcscmp*() as appropriate to collating sequence information in the current locale (category *LC_COLLATE*).

The fact that when *n* is 0 *ws1* is permitted to be a null pointer is useful to determine the size of the *ws1* array prior to making the transformation.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcscmp( ), *wcscoll*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wctob — wide-character to single-byte conversion

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wctob(wint_t c);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wctob*() function shall determine whether *c* corresponds to a member of the extended character set whose character representation is a single byte when in the initial shift state.

The behavior of this function shall be affected by the *LC_CTYPE* category of the current locale.

RETURN VALUE

The *wctob*() function shall return EOF if *c* does not correspond to a character with length one in the initial shift state. Otherwise, it shall return the single-byte representation of that character as an **unsigned char** converted to **int**.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

btowc( )

The Base Definitions volume of POSIX.1-2017, **<stdio.h>**, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wctomb — convert a wide-character code to a character

SYNOPSIS

#include <stdlib.h>

int wctomb(char **s*, wchar_t *wchar*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wctomb*() function shall determine the number of bytes needed to represent the character corresponding to the wide-character code whose value is *wchar* (including any change in the shift state). It shall store the character representation (possibly multiple bytes and any special bytes to change shift state) in the array object pointed to by *s* (if *s* is not a null pointer). At most {MB_CUR_MAX} bytes shall be stored. If *wchar* is 0, a null byte shall be stored, preceded by any shift sequence needed to restore the initial shift state, and *wctomb*() shall be left in the initial shift state.

The behavior of this function is affected by the *LC_CTYPE* category of the current locale. For a state-dependent encoding, this function shall be placed into its initial state by a call for which its character pointer argument, *s*, is a null pointer. Subsequent calls with *s* as other than a null pointer shall cause the internal state of the function to be altered as necessary. A call with *s* as a null pointer shall cause this function to return a non-zero value if encodings have state dependency, and 0 otherwise. Changing the *LC_CTYPE* category causes the shift state of this function to be unspecified.

The *wctomb*() function need not be thread-safe.

The implementation shall behave as if no function defined in this volume of POSIX.1-2017 calls *wctomb*().

RETURN VALUE

If *s* is a null pointer, *wctomb*() shall return a non-zero or 0 value, if character encodings, respectively, do or do not have state-dependent encodings. If *s* is not a null pointer, *wctomb*() shall return −1 if the value of *wchar* does not correspond to a valid character, or return the number of bytes that constitute the character corresponding to the value of *wchar*.

In no case shall the value returned be greater than the value of the {MB_CUR_MAX} macro.

ERRORS

The *wctomb*() function shall fail if:

EILSEQ
An invalid wide-character code is detected.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

mblen( ), *mbtowc*( ), *mbstowcs*( ), *wcstombs*( )

The Base Definitions volume of POSIX.1-2017, **<stdlib.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wctrans, wctrans_l — define character mapping

SYNOPSIS

```
#include <wctype.h>

wctrans_t wctrans(const char *charclass);
wctrans_t wctrans_l(const char *charclass, locale_t locale);
```

DESCRIPTION

For *wctrans*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wctrans*() and *wctrans_l*() functions are defined for valid character mapping names identified in the current locale. The *charclass* is a string identifying a generic character mapping name for which codeset-specific information is required. The following character mapping names are defined in all locales: **tolower** and **toupper**.

These functions shall return a value of type **wctrans_t**, which can be used as the second argument to subsequent calls of *towctrans*() and *towctrans_l*().

The *wctrans*() and *wctrans_l*() functions shall determine values of **wctrans_t** according to the rules of the coded character set defined by character mapping information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*).

The values returned by *wctrans*() shall be valid until a call to *setlocale*() that modifies the category *LC_CTYPE*.

The values returned by *wctrans_l*() shall be valid only in calls to *towctrans_l*() with a locale represented by *locale* with the same *LC_CTYPE* category value.

The behavior is undefined if the *locale* argument to *wctrans_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *wctrans*() and *wctrans_l*() functions shall return 0 and may set *errno* to indicate the error if the given character mapping name is not valid for the current locale (category *LC_CTYPE*); otherwise, they shall return a non-zero object of type **wctrans_t** that can be used in calls to *towctrans*() and *towctrans_l*().

ERRORS

These functions may fail if:

EINVAL

The character mapping name pointed to by *charclass* is not valid in the current locale.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

towctrans( )

The Base Definitions volume of POSIX.1-2017, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wctype, wctype_l — define character class

SYNOPSIS

```
#include <wctype.h>

wctype_t wctype(const char *property);
wctype_t wctype_l(const char *property, locale_t locale);
```

DESCRIPTION

For *wctype*(): The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wctype*() and *wctype_l*() functions are defined for valid character class names as defined in the current locale or in the locale represented by *locale*, respectively.

The *property* argument is a string identifying a generic character class for which codeset-specific type information is required. The following character class names shall be defined in all locales:

```
tab(!); lB lB lB.  T{
alnum
alpha
blank
cntrl
T}!T{
digit
graph
lower
print
T}!T{
punct
space
upper
xdigit
T}
```

Additional character class names defined in the locale definition file (category *LC_CTYPE*) can also be specified.

These functions shall return a value of type **wctype_t**, which can be used as the second argument to subsequent calls of *iswctype*() and *iswctype_l*().

The *wctype*() and *wctype_l*() functions shall determine values of **wctype_t** according to the rules of the coded character set defined by character type information in the current locale or in the locale represented by *locale*, respectively (category *LC_CTYPE*).

The values returned by *wctype*() shall be valid until a call to *setlocale*() that modifies the category *LC_CTYPE*.

The values returned by *wctype_l*() shall be valid only in calls to *iswctype_l*() with a locale represented by *locale* with the same *LC_CTYPE* category value.

The behavior is undefined if the *locale* argument to *wctype_l*() is the special locale object LC_GLOBAL_LOCALE or is not a valid locale object handle.

RETURN VALUE

The *wctype*() and *wctype_l*() functions shall return 0 if the given character class name is not valid for the current locale (category *LC_CTYPE*); otherwise, they shall return an object of type **wctype_t** that can be used in calls to *iswctype*() and *iswctype_l*().

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

iswctype( )

The Base Definitions volume of POSIX.1-2017, **<wctype.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wcwidth — number of column positions of a wide-character code

SYNOPSIS

```
#include <wchar.h>

int wcwidth(wchar_t wc);
```

DESCRIPTION

The *wcwidth*() function shall determine the number of column positions required for the wide character *wc*. The application shall ensure that the value of *wc* is a character representable as a **wchar_t**, and is a wide-character code corresponding to a valid character in the current locale.

RETURN VALUE

The *wcwidth*() function shall either return 0 (if *wc* is a null wide-character code), or return the number of column positions to be occupied by the wide-character code *wc*, or return −1 (if *wc* does not correspond to a printable wide-character code).

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

This function was removed from the final ISO/IEC 9899: 1990/Amendment 1: 1995 (E), and the return value for a non-printable wide character is not specified.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wcswidth( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wmemchr — find a wide character in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemchr(const wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wmemchr*() function shall locate the first occurrence of *wc* in the initial *n* wide characters of the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws* is a valid pointer and the function behaves as if no valid occurrence of *wc* is found.

RETURN VALUE

The *wmemchr*() function shall return a pointer to the located wide character, or a null pointer if the wide character does not occur in the object.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemcmp( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wmemcmp — compare wide characters in memory

SYNOPSIS

#include <wchar.h>

int wmemcmp(const wchar_t **ws1*, const wchar_t **ws2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wmemcmp*() function shall compare the first *n* wide characters of the object pointed to by *ws1* to the first *n* wide characters of the object pointed to by *ws2*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall behave as if the two objects compare equal.

RETURN VALUE

The *wmemcmp*() function shall return an integer greater than, equal to, or less than zero, respectively, as the object pointed to by *ws1* is greater than, equal to, or less than the object pointed to by *ws2*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcpy*( ), *wmemmove*( ), *wmemset*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wmemcpy — copy wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemcpy(wchar_t *restrict ws1, const wchar_t *restrict ws2,
   size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wmemcpy*() function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemcpy*() function shall return the value of *ws1*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemmove*( ), *wmemset*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wmemmove — copy wide characters in memory with overlapping areas

SYNOPSIS

#include <wchar.h>

wchar_t *wmemmove(wchar_t **ws1*, const wchar_t **ws2*, size_t *n*);

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wmemmove*() function shall copy *n* wide characters from the object pointed to by *ws2* to the object pointed to by *ws1*. Copying shall take place as if the *n* wide characters from the object pointed to by *ws2* are first copied into a temporary array of *n* wide characters that does not overlap the objects pointed to by *ws1* or *ws2*, and then the *n* wide characters from the temporary array are copied into the object pointed to by *ws1*.

This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws1* and *ws2* are valid pointers, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemmove*() function shall return the value of *ws1*.

ERRORS

No errors are defined

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemset*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see

https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wmemset — set wide characters in memory

SYNOPSIS

```
#include <wchar.h>

wchar_t *wmemset(wchar_t *ws, wchar_t wc, size_t n);
```

DESCRIPTION

The functionality described on this reference page is aligned with the ISO C standard. Any conflict between the requirements described here and the ISO C standard is unintentional. This volume of POSIX.1-2017 defers to the ISO C standard.

The *wmemset*() function shall copy the value of *wc* into each of the first *n* wide characters of the object pointed to by *ws*. This function shall not be affected by locale and all **wchar_t** values shall be treated identically. The null wide character and **wchar_t** values not corresponding to valid characters shall not be treated specially.

If *n* is zero, the application shall ensure that *ws* is a valid pointer, and the function shall copy zero wide characters.

RETURN VALUE

The *wmemset*() functions shall return the value of *ws*.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

None.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

wmemchr( ), *wmemcmp*( ), *wmemcpy*( ), *wmemmove*( )

The Base Definitions volume of POSIX.1-2017, **<wchar.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wordexp, wordfree — perform word expansions

SYNOPSIS

```
#include <wordexp.h>

int wordexp(const char *restrict words, wordexp_t *restrict pwordexp,
   int flags);
void wordfree(wordexp_t *pwordexp);
```

DESCRIPTION

The *wordexp*() function shall perform word expansions as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.6*, *Word Expansions*, subject to quoting as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.2*, *Quoting*, and place the list of expanded words into the structure pointed to by *pwordexp*.

The *words* argument is a pointer to a string containing one or more words to be expanded. The expansions shall be the same as would be performed by the command line interpreter if *words* were the part of a command line representing the arguments to a utility. Therefore, the application shall ensure that *words* does not contain an unquoted <newline> character or any of the unquoted shell special characters **'|'**, **'&'**, **';'**, **'<'**, **'>'** except in the context of command substitution as specified in the Shell and Utilities volume of POSIX.1-2017, *Section 2.6.3*, *Command Substitution*. It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. The application shall ensure that every member of *words* which it expects to have expanded by *wordexp*() does not contain an unquoted initial comment character. The application shall also ensure that any words which it intends to be ignored (because they begin or continue a comment) are deleted from *words*. If the argument *words* contains an unquoted comment character (<number-sign>) that is the beginning of a token, *wordexp*() shall either treat the comment character as a regular character, or interpret it as a comment indicator and ignore the remainder of *words*.

The structure type **wordexp_t** is defined in the *<wordexp.h>* header and includes at least the following members:

center box tab(!); cB | cB | cB lw(1.25i)B | lw(1.25i)I | lw(2.5i). Member Type!Member Name!Description _ size_t!we_wordc!Count of words matched by *words*. char **!we_wordv!Pointer to list of expanded words. size_t!we_offs!T{ Slots to reserve at the beginning of *pwordexp*−>*we_wordv*. T}

The *wordexp*() function shall store the number of generated words into *pwordexp*−>*we_wordc* and a pointer to a list of pointers to words in *pwordexp*−>*we_wordv*. Each individual field created during field splitting (see the Shell and Utilities volume of POSIX.1-2017, *Section 2.6.5*, *Field Splitting*) or pathname expansion (see the Shell and Utilities volume of POSIX.1-2017, *Section 2.6.6*, *Pathname Expansion*) shall be a separate word in the *pwordexp*−>*we_wordv* list. The words shall be in order as described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.6*, *Word Expansions*. The first pointer after the last word pointer shall be a null pointer. The expansion of special parameters described in the Shell and Utilities volume of POSIX.1-2017, *Section 2.5.2*, *Special Parameters* is unspecified.

It is the caller's responsibility to allocate the storage pointed to by *pwordexp*. The *wordexp*() function shall allocate other space as needed, including memory pointed to by *pwordexp*−>*we_wordv*. The *wordfree*() function frees any memory associated with *pwordexp* from a previous call to *wordexp*().

The *flags* argument is used to control the behavior of *wordexp*(). The value of *flags* is the bitwise-inclusive OR of zero or more of the following constants, which are defined in *<wordexp.h>*:

WRDE_APPEND
: Append words generated to the ones from a previous call to *wordexp*().

WRDE_DOOFFS
: Make use of *pwordexp*−>*we_offs*. If this flag is set, *pwordexp*−>*we_offs* is used to specify how many null pointers to add to the beginning of *pwordexp*−>*we_wordv*. In other words, *pwordexp*−>*we_wordv* shall point to *pwordexp*−>*we_offs* null pointers, followed by *pwordexp*−>*we_wordc* word pointers, followed by a null pointer.

WRDE_NOCMD
: If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2017, fail if command substitution, as specified in the Shell and Utilities volume of POSIX.1-2017, *Section 2.6.3*, *Command Substitution*, is requested.

WRDE_REUSE
: The *pwordexp* argument was passed to a previous successful call to *wordexp*(), and has not been passed to *wordfree*(). The result shall be the same as if the application had called *wordfree*() and then called *wordexp*() without WRDE_REUSE.

WRDE_SHOWERR
: Do not redirect *stderr* to **/dev/null**.

WRDE_UNDEF
: Report error on an attempt to expand an undefined shell variable.

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to *wordexp*(). The following rules apply to applications when two or more calls to *wordexp*() are made with the same value of *pwordexp* and without intervening calls to *wordfree*():

1. The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.
2. All of the calls shall set WRDE_DOOFFS, or all shall not set it.
3. After the second and each subsequent call, *pwordexp*−>*we_wordv* shall point to a list containing the following:
   a. Zero or more null pointers, as specified by WRDE_DOOFFS and *pwordexp*−>*we_offs*
   b. Pointers to the words that were in the *pwordexp*−>*we_wordv* list before the call, in the same order as before
   c. Pointers to the new words generated by the latest call, in the specified order
4. The count returned in *pwordexp*−>*we_wordc* shall be the total number of words from all of the calls.
5. The application can change any of the fields after a call to *wordexp*(), but if it does it shall reset them to the original value before a subsequent call, using the same *pwordexp* value, to *wordfree*() or *wordexp*() with the WRDE_APPEND or WRDE_REUSE flag.

If the implementation supports the utilities defined in the Shell and Utilities volume of POSIX.1-2017, and *words* contains an unquoted character—<newline>, **'|'**, **'&'**, **';'**, **'<'**, **'>'**, **'('**, **')'**, **'{'**, **'}'**—in an inappropriate context, *wordexp*() shall fail, and the number of expanded words shall be 0.

Unless WRDE_SHOWERR is set in *flags*, *wordexp*() shall redirect *stderr* to **/dev/null** for any utilities executed as a result of command substitution while expanding *words*. If WRDE_SHOWERR is set, *wordexp*() may write messages to *stderr* if syntax errors are detected while expanding *words*, unless the *stderr* stream has wide orientation in which case the behavior is undefined. It is unspecified whether any write errors encountered while outputting such messages will affect the *stderr* error indicator or the value of *errno*.

The application shall ensure that if WRDE_DOOFFS is set, then *pwordexp*−>*we_offs* has the same value for each *wordexp*() call and *wordfree*() call using a given *pwordexp*.

The results are unspecified if WRDE_APPEND and WRDE_REUSE are both specified.

The following constants are defined as error return values:

WRDE_BADCHAR
: One of the unquoted characters—<newline>, **'|'**, **'&'**, **';'**, **'<'**, **'>'**, **'('**, **')'**, **'{'**, **'}'**—appears in *words* in an inappropriate context.

WRDE_BADVAL
: Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.

WRDE_CMDSUB
: Command substitution requested when WRDE_NOCMD was set in *flags*.

WRDE_NOSPACE
: Attempt to allocate memory failed.

WRDE_SYNTAX
: Shell syntax error, such as unbalanced parentheses or unterminated string.

RETURN VALUE

Upon successful completion, *wordexp*() shall return 0. Otherwise, a non-zero value, as described in *<wordexp.h>*, shall be returned to indicate an error. If *wordexp*() returns the value WRDE_NOSPACE, then *pwordexp*−>*we_wordc* and *pwordexp*−>*we_wordv* shall be updated to reflect any words that were successfully expanded. In other error cases, if the WRDE_APPEND flag was specified, *pwordexp*->*we_wordc* and *pwordexp*->*we_wordv* shall not be modified.

The *wordfree*() function shall not return a value.

ERRORS

No errors are defined.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

The *wordexp*() function is intended to be used by an application that wants to do all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a pathname (or list of pathnames) and then uses *wordexp*() to process the input, the user could respond with anything that would be valid as input to the shell.

The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side-effects, such as executing a command or writing a file.

POSIX.1-2008 does not require the *wordexp*() function to be thread-safe if passed an expression referencing an environment variable while any other thread is concurrently modifying any environment variable; see *exec*.

Even though the WRDE_SHOWERR flag allows the implementation to write messages to *stderr* during command substitution or syntax errors, this standard does not provide any way to detect write failures during the output of such messages.

Applications which use wide-character output functions with *stderr* should ensure that any calls to *wordexp*() do not write to *stderr*, by avoiding use of the WRDE_SHOWERR flag.

RATIONALE

This function was included as an alternative to *glob*(). There had been continuing controversy over exactly what features should be included in *glob*(). It is hoped that by providing *wordexp*() (which provides all of the shell word expansions, but which may be slow to execute) and *glob*() (which is faster, but which only performs pathname expansion, without tilde or parameter expansion) this will satisfy the majority of applications.

While *wordexp*() could be implemented entirely as a library routine, it is expected that most implementations run a shell in a subprocess to do the expansion.

Two different approaches have been proposed for how the required information might be presented to the shell and the results returned. They are presented here as examples.

One proposal is to extend the *echo* utility by adding a **−q** option. This option would cause *echo* to add a <backslash> before each <backslash> and <blank> that occurs within an argument. The *wordexp*() function could then invoke the shell as follows:

```
(void) strcpy(buffer, "echo -q");
(void) strcat(buffer, words);
if ((flags & WRDE_SHOWERR) == 0)
    (void) strcat(buffer, "2>/dev/null");
f = popen(buffer, "r");
```

The *wordexp*() function would read the resulting output, remove unquoted <backslash> characters, and break into words at unquoted <blank> characters. If the WRDE_NOCMD flag was set, *wordexp*() would have to scan *words* before starting the subshell to make sure that there would be no command substitution. In any case, it would have to scan *words* for unquoted special characters.

Another proposal is to add the following options to *sh*:

−w *wordlist*

This option provides a wordlist expansion service to applications. The words in *wordlist* shall be expanded and the following written to standard output:

1. The count of the number of words after expansion, in decimal, followed by a null byte
2. The number of bytes needed to represent the expanded words (not including null separators), in decimal, followed by a null byte
3. The expanded words, each terminated by a null byte

If an error is encountered during word expansion, *sh* exits with a non-zero status after writing the former to report any words successfully expanded

−P Run in ‘‘protected’’ mode. If specified with the **−w** option, no command substitution shall be performed.

With these options, *wordexp*() could be implemented fairly simply by creating a subprocess using *fork*() and executing *sh* using the line:

```
execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

after directing standard error to **/dev/null**.

It seemed objectionable for a library routine to write messages to standard error, unless explicitly requested, so *wordexp*() is required to redirect standard error to **/dev/null** to ensure that no messages are generated, even for commands executed for command substitution. The WRDE_SHOWERR flag can be specified to request that error messages be written.

The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and reallocating memory, if that is possible. A minimal implementation can call *wordfree*() when WRDE_REUSE is set.

FUTURE DIRECTIONS

None.

SEE ALSO

exec, *fnmatch*( ), *glob*( )

The Base Definitions volume of POSIX.1-2017, **<wordexp.h>**

The Shell and Utilities volume of POSIX.1-2017, *Chapter 2*, *Shell Command Language*

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The

original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wprintf — print formatted wide-character output

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wprintf(const wchar_t *restrict format, ...);
```

DESCRIPTION

Refer to *fwprintf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

pwrite, write — write on a file

SYNOPSIS

```
#include <unistd.h>

ssize_t pwrite(int fildes, const void *buf, size_t nbyte,
   off_t offset);
ssize_t write(int fildes, const void *buf, size_t nbyte);
```

DESCRIPTION

The *write*() function shall attempt to write *nbyte* bytes from the buffer pointed to by *buf* to the file associated with the open file descriptor, *fildes*.

Before any action described below is taken, and if *nbyte* is zero and the file is a regular file, the *write*() function may detect and return errors as described below. In the absence of errors, or if error detection is not performed, the *write*() function shall return zero and have no other results. If *nbyte* is zero and the file is not a regular file, the results are unspecified.

On a regular file or other file capable of seeking, the actual writing of data shall proceed from the position in the file indicated by the file offset associated with *fildes*. Before successful return from *write*(), the file offset shall be incremented by the number of bytes actually written. On a regular file, if the position of the last byte written is greater than or equal to the length of the file, the length of the file shall be set to this position plus one.

On a file not capable of seeking, writing shall always take place starting at the current position. The value of a file offset associated with such a device is undefined.

If the O_APPEND flag of the file status flags is set, the file offset shall be set to the end of the file prior to each write and no intervening file modification operation shall occur between changing the file offset and the write operation.

If a *write*() requests that more bytes be written than there is room for (for example, the file size limit of the process or the physical end of a medium), only as many bytes as there is room for shall be written. For example, suppose there is space for 20 bytes more in a file before reaching a limit. A write of 512 bytes will return 20. The next write of a non-zero number of bytes would give a failure return (except as noted below).

If the request would cause the file size to exceed the soft file size limit for the process and there is no room for any bytes to be written, the request shall fail and the implementation shall generate the SIGXFSZ signal for the thread.

If *write*() is interrupted by a signal before it writes any data, it shall return −1 with *errno* set to **[EINTR]**.

If *write*() is interrupted by a signal after it successfully writes some data, it shall return the number of bytes written.

If the value of *nbyte* is greater than {SSIZE_MAX}, the result is implementation-defined.

After a *write*() to a regular file has successfully returned:

* Any successful *read*() from each byte position in the file that was modified by that write shall return the data specified by the *write*() for that position until such byte positions are again modified.
* Any subsequent successful *write*() to the same byte position in the file shall overwrite that file data.

Write requests to a pipe or FIFO shall be handled in the same way as a regular file with the following exceptions:

* There is no file offset associated with a pipe, hence each write request shall append to the end of the pipe.

* Write requests of {PIPE_BUF} bytes or less shall not be interleaved with data from other processes doing writes on the same pipe. Writes of greater than {PIPE_BUF} bytes may have data interleaved, on arbitrary boundaries, with writes by other processes, whether or not the O_NONBLOCK flag of the file status flags is set.
* If the O_NONBLOCK flag is clear, a write request may cause the thread to block, but on normal completion it shall return *nbyte*.
* If the O_NONBLOCK flag is set, *write*() requests shall be handled differently, in the following ways:
  -- The *write*() function shall not block the thread.
  -- A write request for {PIPE_BUF} or fewer bytes shall have the following effect: if there is sufficient space available in the pipe, *write*() shall transfer all the data and return the number of bytes requested. Otherwise, *write*() shall transfer no data and return −1 with *errno* set to **[EAGAIN]**.
  -- A write request for more than {PIPE_BUF} bytes shall cause one of the following:
    -- When at least one byte can be written, transfer what it can and return the number of bytes written. When all data previously written to the pipe is read, it shall transfer at least {PIPE_BUF} bytes.
    -- When no data can be written, transfer no data, and return −1 with *errno* set to **[EAGAIN]**.

When attempting to write to a file descriptor (other than a pipe or FIFO) that supports non-blocking writes and cannot accept the data immediately:

* If the O_NONBLOCK flag is clear, *write*() shall block the calling thread until the data can be accepted.
* If the O_NONBLOCK flag is set, *write*() shall not block the thread. If some data can be written without blocking the thread, *write*() shall write what it can and return the number of bytes written. Otherwise, it shall return −1 and set *errno* to **[EAGAIN]**.

Upon successful completion, where *nbyte* is greater than 0, *write*() shall mark for update the last data modification and last file status change timestamps of the file, and if the file is a regular file, the S_ISUID and S_ISGID bits of the file mode may be cleared.

For regular files, no data transfer shall occur past the offset maximum established in the open file description associated with *fildes*.

If *fildes* refers to a socket, *write*() shall be equivalent to *send*() with no flags set.

If the O_DSYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O data integrity completion.

If the O_SYNC bit has been set, write I/O operations on the file descriptor shall complete as defined by synchronized I/O file integrity completion.

If *fildes* refers to a shared memory object, the result of the *write*() function is unspecified.

If *fildes* refers to a typed memory object, the result of the *write*() function is unspecified.

If *fildes* refers to a STREAM, the operation of *write*() shall be determined by the values of the minimum and maximum *nbyte* range (packet size) accepted by the STREAM. These values are determined by the topmost STREAM module. If *nbyte* falls within the packet size range, *nbyte* bytes shall be written. If *nbyte* does not fall within the range and the minimum packet size value is 0, *write*() shall break the buffer into maximum packet size segments prior to sending the data downstream (the last segment may contain less than the maximum packet size). If *nbyte* does not fall within the range and the minimum value is non-zero, *write*() shall fail with *errno* set to **[ERANGE]**. Writing a zero-length buffer (*nbyte* is 0) to a STREAMS device sends 0 bytes with 0 returned. However, writing a zero-length buffer to a STREAMS-based pipe or FIFO sends no message and 0 is returned. The process may issue I_SWROPT *ioctl*() to enable zero-length messages to be sent across the pipe or FIFO.

When writing to a STREAM, data messages are created with a priority band of 0. When writing to a STREAM that is not a pipe or FIFO:

* If O_NONBLOCK is clear, and the STREAM cannot accept data (the STREAM write queue is full due to internal flow control conditions), *write*() shall block until data can be accepted.
* If O_NONBLOCK is set and the STREAM cannot accept data, *write*() shall return −1 and set *errno* to **[EAGAIN]**.
* If O_NONBLOCK is set and part of the buffer has been written while a condition in which the STREAM cannot accept additional data occurs, *write*() shall terminate and return the number of bytes written.

In addition, *write*() shall fail if the STREAM head has processed an asynchronous error before the call. In this case, the value of *errno* does not reflect the result of *write*(), but reflects the prior error.

The *pwrite*() function shall be equivalent to *write*(), except that it writes into a given position and does not change the file offset (regardless of whether O_APPEND is set). The first three arguments to *pwrite*() are the same as *write*() with the addition of a fourth argument *offset* for the desired position inside the file. An attempt to perform a *pwrite*() on a file that is incapable of seeking shall result in an error.

RETURN VALUE

Upon successful completion, these functions shall return the number of bytes actually written to the file associated with *fildes*. This number shall never be greater than *nbyte*. Otherwise, −1 shall be returned and *errno* set to indicate the error.

ERRORS

These functions shall fail if:

EAGAIN
The file is neither a pipe, nor a FIFO, nor a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the *write*() operation.

EBADF
The *fildes* argument is not a valid file descriptor open for writing.

EFBIG
An attempt was made to write a file that exceeds the implementation-defined maximum file size or the file size limit of the process, and there was no room for any bytes to be written.

EFBIG
The file is a regular file, *nbyte* is greater than 0, and the starting position is greater than or equal to the offset maximum established in the open file description associated with *fildes*.

EINTR
The write operation was terminated due to the receipt of a signal, and no data was transferred.

EIO The process is a member of a background process group attempting to write to its controlling terminal, TOSTOP is set, the calling thread is not blocking SIGTTOU, the process is not ignoring SIGTTOU, and the process group of the process is orphaned. This error may also be returned under implementation-defined conditions.

ENOSPC
There was no free space remaining on the device containing the file.

ERANGE
The transfer request size was outside the range supported by the STREAMS file associated with *fildes*.

The *pwrite*() function shall fail if:

EINVAL
The file is a regular file or block special file, and the *offset* argument is negative. The file offset shall remain unchanged.

ESPIPE
The file is incapable of seeking.

The *write*() function shall fail if:

EAGAIN
The file is a pipe or FIFO, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write operation.

EAGAIN or **EWOULDBLOCK**
The file is a socket, the O_NONBLOCK flag is set for the file descriptor, and the thread would be delayed in the write operation.

ECONNRESET
A write was attempted on a socket that is not connected.

EPIPE An attempt is made to write to a pipe or FIFO that is not open for reading by any process, or that only has one end open. A SIGPIPE signal shall also be sent to the thread.

EPIPE A write was attempted on a socket that is shut down for writing, or is no longer connected. In the latter case, if the socket is of type SOCK_STREAM, a SIGPIPE signal shall also be sent to the thread.

These functions may fail if:

EINVAL
The STREAM or multiplexer referenced by *fildes* is linked (directly or indirectly) downstream from a multiplexer.

EIO A physical I/O error has occurred.

ENOBUFS
Insufficient resources were available in the system to perform the operation.

ENXIO
A request was made of a nonexistent device, or the request was outside the capabilities of the device.

ENXIO
A hangup occurred on the STREAM being written to.

A write to a STREAMS file may fail if an error message has been received at the STREAM head. In this case, *errno* is set to the value included in the error message.

The *write*() function may fail if:

EACCES
A write was attempted on a socket and the calling process does not have appropriate privileges.

ENETDOWN
A write was attempted on a socket and the local network interface used to reach the destination is down.

ENETUNREACH
A write was attempted on a socket and no route to the network is present.

The following sections are informative.

EXAMPLES

Writing from a Buffer

The following example writes data from the buffer pointed to by *buf* to the file associated with the file descriptor *fd*.

```
#include <sys/types.h>
#include <string.h>
...
char buf[20];
```

```
size_t nbytes;
ssize_t bytes_written;
int fd;
...
strcpy(buf, "This is a test\n");
nbytes = strlen(buf);

bytes_written = write(fd, buf, nbytes);
...
```

APPLICATION USAGE

None.

RATIONALE

See also the RATIONALE section in *read*().

An attempt to write to a pipe or FIFO has several major characteristics:

* *Atomic/non-atomic*: A write is atomic if the whole amount written in one operation is not interleaved with data from any other process. This is useful when there are multiple writers sending data to a single reader. Applications need to know how large a write request can be expected to be performed atomically. This maximum is called {PIPE_BUF}. This volume of POSIX.1-2017 does not say whether write requests for more than {PIPE_BUF} bytes are atomic, but requires that writes of {PIPE_BUF} or fewer bytes shall be atomic.

* *Blocking/immediate*: Blocking is only possible with O_NONBLOCK clear. If there is enough space for all the data requested to be written immediately, the implementation should do so. Otherwise, the calling thread may block; that is, pause until enough space is available for writing. The effective size of a pipe or FIFO (the maximum amount that can be written in one operation without blocking) may vary dynamically, depending on the implementation, so it is not possible to specify a fixed value for it.

* *Complete/partial/deferred*: A write request:

  ```
  int fildes;
  size_t nbyte;
  ssize_t ret;
  char *buf;

  ret = write(fildes, buf, nbyte);
  ```

  may return:

  Complete *ret*=*nbyte*

  Partial *ret*<*nbyte*

  This shall never happen if *nbyte*≤{PIPE_BUF}. If it does happen (with *nbyte*>{PIPE_BUF}), this volume of POSIX.1-2017 does not guarantee atomicity, even if *ret*≤{PIPE_BUF}, because atomicity is guaranteed according to the amount *requested*, not the amount *written*.

  Deferred: *ret*=−1, *errno*=[EAGAIN]

  This error indicates that a later request may succeed. It does not indicate that it *shall* succeed, even if *nbyte*≤{PIPE_BUF}, because if no process reads from the pipe or FIFO, the write never succeeds. An application could usefully count the number of times **[EAGAIN]** is caused by a particular value of *nbyte*>{PIPE_BUF} and perhaps do later writes with a smaller value, on the assumption that the effective size of the pipe may have decreased.

  Partial and deferred writes are only possible with O_NONBLOCK set.

The relations of these properties are shown in the following tables:

center box tab(!); cB s s s cB | cB cB c l1 | lw(1.25i)1 lw(1.25i)1 lw(1.25i). Write to a Pipe or FIFO with O_NONBLOCK *clear* _ Immediately Writable:!None!Some!*nbyte* _ *nbyte*≤{PIPE_BUF}!Atomic blocking!Atomic blocking!Atomic immediate !*nbyte*!*nbyte*!*nbyte* _ *nbyte*>{PIPE_BUF}!Blocking *nbyte*!Blocking *nbyte*!Blocking *nbyte*

If the O_NONBLOCK flag is clear, a write request shall block if the amount writable immediately is less than that requested. If the flag is set (by *fcntl*()), a write request shall never block.

center box tab(!); cB s s s cB | cB cB c l1 | lw(1.25i)1 lw(1.25i)1 lw(1.25i). Write to a Pipe or FIFO with O_NONBLOCK *set* _ Immediately Writable:!None!Some!*nbyte* _ *nbyte*≤{PIPE_BUF}!−1, [EAGAIN]!−1, [EAGAIN]!Atomic *nbyte* _ *nbyte*>{PIPE_BUF}!−1, [EAGAIN]!<*nbyte* or −1,!≤*nbyte* or −1, !![EAGAIN]![EAGAIN]

There is no exception regarding partial writes when O_NONBLOCK is set. With the exception of writing to an empty pipe, this volume of POSIX.1-2017 does not specify exactly when a partial write is performed since that would require specifying internal details of the implementation. Every application should be prepared to handle partial writes when O_NONBLOCK is set and the requested amount is greater than {PIPE_BUF}, just as every application should be prepared to handle partial writes on other kinds of file descriptors.

The intent of forcing writing at least one byte if any can be written is to assure that each write makes progress if there is any room in the pipe. If the pipe is empty, {PIPE_BUF} bytes must be written; if not, at least some progress must have been made.

Where this volume of POSIX.1-2017 requires −1 to be returned and *errno* set to **[EAGAIN]**, most historical implementations return zero (with the O_NDELAY flag set, which is the historical predecessor of O_NONBLOCK, but is not itself in this volume of POSIX.1-2017). The error indications in this volume of POSIX.1-2017 were chosen so that an application can distinguish these cases from end-of-file. While *write*() cannot receive an indication of end-of-file, *read*() can, and the two functions have similar return values. Also, some existing systems (for example, Eighth Edition) permit a write of zero bytes to mean that the reader should get an end-of-file indication; for those systems, a return value of zero from *write*() indicates a successful write of an end-of-file indication.

Implementations are allowed, but not required, to perform error checking for *write*() requests of zero bytes.

The concept of a {PIPE_MAX} limit (indicating the maximum number of bytes that can be written to a pipe in a single operation) was considered, but rejected, because this concept would unnecessarily limit application writing.

See also the discussion of O_NONBLOCK in *read*().

Writes can be serialized with respect to other reads and writes. If a *read*() of file data can be proven (by any means) to occur after a *write*() of the data, it must reflect that *write*(), even if the calls are made by different processes. A similar requirement applies to multiple write operations to the same file position. This is needed to guarantee the propagation of data from *write*() calls to subsequent *read*() calls. This requirement is particularly significant for networked file systems, where some caching schemes violate these semantics.

Note that this is specified in terms of *read*() and *write*(). The XSI extensions *readv*() and *writev*() also obey these semantics. A new "high-performance" write analog that did not follow these serialization requirements would also be permitted by this wording. This volume of POSIX.1-2017 is also silent about any effects of application-level caching (such as that done by *stdio*).

This volume of POSIX.1-2017 does not specify the value of the file offset after an error is returned; there are too many cases. For programming errors, such as **[EBADF]**, the concept is meaningless since no file is involved. For errors that are detected immediately, such as **[EAGAIN]**, clearly the pointer should not change. After an interrupt or hardware error, however, an updated value would be very useful and is the behavior of many implementations.

This volume of POSIX.1-2017 does not specify the behavior of concurrent writes to a regular file from multiple threads, except that each write is atomic (see *Section 2.9.7*, *Thread Interactions with Regular File Operations*). Applications should use some form of concurrency control.

This volume of POSIX.1-2017 intentionally does not specify any *pwrite*() errors related to pipes, FIFOs, and sockets other than **[ESPIPE]**.

FUTURE DIRECTIONS

None.

SEE ALSO

chmod( ), *creat*( ), *dup*( ), *fcntl*( ), *getrlimit*( ), *lseek*( ), *open*( ), *pipe*( ), *read*( ), *ulimit*( ), *writev*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<stropts.h>**, **<sys_uio.h>**, **<unistd.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

writev — write a vector

SYNOPSIS

#include <sys/uio.h>

ssize_t writev(int *fildes*, const struct iovec **iov*, int *iovcnt*);

DESCRIPTION

The *writev*() function shall be equivalent to *write*(), except as described below. The *writev*() function shall gather output data from the *iovcnt* buffers specified by the members of the *iov* array: *iov*[0], *iov*[1], . . ., *iov*[*iovcnt*−1]. The *iovcnt* argument is valid if greater than 0 and less than or equal to {IOV_MAX}, as defined in *<limits.h>*.

Each *iovec* entry specifies the base address and length of an area in memory from which data should be written. The *writev*() function shall always write a complete area before proceeding to the next.

If *fildes* refers to a regular file and all of the *iov_len* members in the array pointed to by *iov* are 0, *writev*() shall return 0 and have no other effect. For other file types, the behavior is unspecified.

If the sum of the *iov_len* values is greater than {SSIZE_MAX}, the operation shall fail and no data shall be transferred.

RETURN VALUE

Upon successful completion, *writev*() shall return the number of bytes actually written. Otherwise, it shall return a value of −1, the file-pointer shall remain unchanged, and *errno* shall be set to indicate an error.

ERRORS

Refer to *write*( ).

In addition, the *writev*() function shall fail if:

EINVAL
The sum of the *iov_len* values in the *iov* array would overflow an **ssize_t**.

The *writev*() function may fail and set *errno* to:

EINVAL
The *iovcnt* argument was less than or equal to 0, or greater than {IOV_MAX}.

The following sections are informative.

EXAMPLES

Writing Data from an Array

The following example writes data from the buffers specified by members of the *iov* array to the file associated with the file descriptor *fd*.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
...
ssize_t bytes_written;
int fd;
char *buf0 = "short string\n";
char *buf1 = "This is a longer string\n";
char *buf2 = "This is the longest string in this example\n";
int iovcnt;
```

```
struct iovec iov[3];

iov[0].iov_base = buf0;
iov[0].iov_len = strlen(buf0);
iov[1].iov_base = buf1;
iov[1].iov_len = strlen(buf1);
iov[2].iov_base = buf2;
iov[2].iov_len = strlen(buf2);
...
iovcnt = sizeof(iov) / sizeof(struct iovec);

bytes_written = writev(fd, iov, iovcnt);
...
```

APPLICATION USAGE

None.

RATIONALE

Refer to *write*( ).

FUTURE DIRECTIONS

None.

SEE ALSO

readv( ), *write*( )

The Base Definitions volume of POSIX.1-2017, **<limits.h>**, **<sys_uio.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

wscanf — convert formatted wide-character input

SYNOPSIS

```
#include <stdio.h>
#include <wchar.h>

int wscanf(const wchar_t *restrict format, ...);
```

DESCRIPTION

Refer to *fwscanf*( ).

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .

PROLOG

This manual page is part of the POSIX Programmer's Manual. The Linux implementation of this interface may differ (consult the corresponding Linux manual page for details of Linux behavior), or the interface may not be implemented on Linux.

NAME

y0, y1, yn — Bessel functions of the second kind

SYNOPSIS

```
#include <math.h>

double y0(double x);
double y1(double x);
double yn(int n, double x);
```

DESCRIPTION

The *y0*(), *y1*(), and *yn*() functions shall compute Bessel functions of *x* of the second kind of orders 0, 1, and *n*, respectively.

An application wishing to check for error situations should set *errno* to zero and call *feclearexcept*(FE_ALL_EXCEPT) before calling these functions. On return, if *errno* is non-zero or *fetestexcept*(FE_INVALID | FE_DIVBYZERO | FE_OVERFLOW | FE_UNDERFLOW) is non-zero, an error has occurred.

RETURN VALUE

Upon successful completion, these functions shall return the relevant Bessel value of *x* of the second kind.

If *x* is NaN, NaN shall be returned.

If the *x* argument to these functions is negative, −HUGE_VAL or NaN shall be returned, and a domain error may occur.

If *x* is 0.0, −HUGE_VAL shall be returned and a pole error may occur.

If the correct result would cause underflow, 0.0 shall be returned and a range error may occur.

If the correct result would cause overflow, −HUGE_VAL or 0.0 shall be returned and a range error may occur.

ERRORS

These functions may fail if:

Domain Error
: The value of *x* is negative.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[EDOM]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the invalid floating-point exception shall be raised.

Pole Error
: The value of *x* is zero.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the divide-by-zero floating-point exception shall be raised.

Range Error
: The correct result would cause overflow.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the overflow floating-point exception shall be raised.

Range Error
: The value of *x* is too large in magnitude, or the correct result would cause underflow.

  If the integer expression (*math_errhandling* & MATH_ERRNO) is non-zero, then *errno* shall be set to **[ERANGE]**. If the integer expression (*math_errhandling* & MATH_ERREXCEPT) is non-zero, then the underflow floating-point exception shall be raised.

The following sections are informative.

EXAMPLES

None.

APPLICATION USAGE

On error, the expressions (*math_errhandling* & MATH_ERRNO) and (*math_errhandling* & MATH_ERREXCEPT) are independent of each other, but at least one of them must be non-zero.

RATIONALE

None.

FUTURE DIRECTIONS

None.

SEE ALSO

feclearexcept( ), *fetestexcept*( ), *isnan*( ), *j0*( )

The Base Definitions volume of POSIX.1-2017, *Section 4.20*, *Treatment of Error Conditions for Mathematical Functions*, **<math.h>**

COPYRIGHT

Portions of this text are reprinted and reproduced in electronic form from IEEE Std 1003.1-2017, Standard for Information Technology -- Portable Operating System Interface (POSIX), The Open Group Base Specifications Issue 7, 2018 Edition, Copyright (C) 2018 by the Institute of Electrical and Electronics Engineers, Inc and The Open Group. In the event of any discrepancy between this version and the original IEEE and The Open Group Standard, the original IEEE and The Open Group Standard is the referee document. The original Standard can be obtained online at http://www.opengroup.org/unix/online.html .

Any typographical or formatting errors that appear in this page are most likely to have been introduced during the conversion of the source files to man page format. To report such errors, see https://www.kernel.org/doc/man-pages/reporting_bugs.html .