## NAME

**pthread_setcancelstate**, **pthread_setcanceltype**, **pthread_testcancel -** set cancelability state

## LIBRARY

POSIX Threads Library (libpthread, -lpthread)

## 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 atomically both sets the calling thread's cancelability state to the indicated *state* and, if *oldstate* is not NULL, returns 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 atomically both sets the calling thread's cancelability type to the indicated *type* and, if *oldtype* is not NULL, returns 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, are PTHREAD_CANCEL_ENABLE and PTHREAD_CANCEL_DEFERRED respectively.

The **pthread_testcancel**() function creates a cancellation point in the calling thread. The **pthread_testcancel**() function has no effect if cancelability is disabled.

### Cancelability States

The cancelability state of a thread determines the action taken upon receipt of a cancellation request. The thread may control cancellation in a number of ways.

Each thread maintains its own "cancelability state" which may be encoded in two bits:

*Cancelability Enable*When cancelability is PTHREAD_CANCEL_DISABLE, cancellation requests against the target thread are held pending.

*Cancelability Type*When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_ASYNCHRONOUS, new or pending cancellation requests may be acted upon at any time. When cancelability is enabled and the cancelability type is PTHREAD_CANCEL_DEFERRED, cancellation requests are held pending until a cancellation point (see below) is reached. If cancelability is disabled, the setting of the cancelability type has no immediate effect as all cancellation requests are held pending; however, once cancelability is enabled again the new type will be in effect.

**Cancellation Points**

Cancellation points will occur when a thread is executing the following functions:

| | |
|---|---|
| **accept**() | |
| **accept4**() | |
| **aio_suspend**() | |
| **connect**() | |
| **clock_nanosleep**() | |
| **close**() | |
| **creat**() | |
| **fcntl**() | The **fcntl**() function is a cancellation point if *cmd* is F_SETLKW. |
| **fdatasync**() | |
| **fsync**() | |
| **kevent**() | The **kevent**() function is a cancellation point if it is potentially blocking, such as when the *nevents* argument is non-zero. |
| **mq_receive**() | |
| **mq_send**() | |
| **mq_timedreceive**() | |
| **mq_timedsend**() | |
| **msync**() | |
| **nanosleep**() | |
| **open**() | |
| **openat**() | |
| **pause**() | |
| **poll**() | |
| **ppoll**() | |
| **pselect**() | |
| **pthread_cond_timedwait**() | |

**pthread_cond_wait**()
**pthread_join**()
**pthread_testcancel**()
**read**()
**readv**()
**recv**()
**recvfrom**()
**recvmsg**()
**select**()
**sem_timedwait**()
**sem_clockwait_np**()
**sem_wait**()
**send**()
**sendmsg**()
**sendto**()
**sigsuspend**()
**sigtimedwait**()
**sigwaitinfo**()
**sigwait**()
**sleep**()
**system**()
**tcdrain**()
**usleep**()
**wait**()
**wait3**()
**wait4**()
**wait6**()
**waitid**()
**waitpid**()
**write**()
**writev**()

## NOTES

The **pthread_setcancelstate**() and **pthread_setcanceltype**() functions are used to control the points at which a thread may be asynchronously canceled. For cancellation control to be usable in modular fashion, some rules must be followed.

For purposes of this discussion, consider an object 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 have to worry about how to clean 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 will be 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 that 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.

## RETURN VALUES

If successful, the **pthread_setcancelstate**() and **pthread_setcanceltype**() functions will return zero. Otherwise, an error number shall be returned to indicate the error.

## ERRORS

The function **pthread_setcancelstate**() may fail with:

[EINVAL] The specified state is not PTHREAD_CANCEL_ENABLE or PTHREAD_CANCEL_DISABLE.

The function **pthread_setcanceltype**() may fail with:

[EINVAL] The specified state is not PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.

## SEE ALSO

pthread_cancel(3)

## STANDARDS

The **pthread_testcancel**() function conforms to ISO/IEC 9945-1:1996 ("POSIX.1"). The standard allows implementations to make many more functions cancellation points.

## AUTHORS

This manual page was written by David Leonard <*d@openbsd.org*> for the OpenBSD implementation

of pthread_cancel(3).