NAME

pthread_join, thr_join – wait for thread termination

SYNOPSIS

POSIX

cc [*flag* ...] *file* ... **–lpthread** [*library* ...]

#include <pthread.h>
int pthread_join(pthread_t *target_thread,* **void** ***status***);**

Solaris

cc [*flag* ...] *file* ... **–lthread** [*library* ...]

#include <thread.h>
int thr_join(thread_t *target_thread***, thread_t** **departed***, void** ***status***);**

MT-LEVEL

MT-Safe

DESCRIPTION

The **pthread_join( )** and **thr_join( )** functions suspend processing of the calling thread until the target *target_thread* completes. *target_thread* must be a member of the current process and it cannot be a detached or daemon thread (see **pthread_create**(3T)).

Several threads cannot wait for the same thread to complete; one thread will complete successfully and the others will terminate with an error of **ESRCH**. **pthread_join( )** or **thr_join( )** will not block processing of the calling thread if the target *target_thread* has already terminated.

pthread_join( ) or **thr_join( )** will return successfully when the target *target_thread* terminates.

POSIX

If a **pthread_join( )** call returns successfully with a non-null *status* argument, the value passed to **pthread_exit**(3T) by the terminating thread will be placed in the location referenced by *status*.

If the **pthread_join( )** calling thread is cancelled, then the target *target_thread* will remain joinable by **pthread_join( )**. However, the calling thread may set up a cancellation cleanup handler on *target_thread* prior to the join call, which may detach the target thread by calling **pthread_detach**(3T). (See **pthread_detach**(3T) and **pthread_cancel**(3T).)

pthread_join( ) does not return the *target_thread*'s ID, as does the Solaris threads' function **thr_join( )**, and it does not cause the calling thread to wait for detached threads. **pthread_join( )** returns **ESRCH** if the target is detached.

Solaris

If a **thr_join( )** call returns successfully with a non-null *status* argument, the value passed to **thr_exit**(3T) by the terminating thread will be placed in the location referenced by *status*.

If the target *target_thread* ID is **0**, **thr_join( )** waits for any undetached thread in the process to terminate.

If *departed* is not **NULL**, it points to a location that is set to the ID of the terminated thread if **thr_join( )** returns successfully.

RETURN VALUES

If successful, both **pthread_join( )** and **thr_join( )** would return **0**; otherwise, an error number is returned to indicate the error.

ERRORS

ESRCH No undetached thread could be found corresponding to that specified by the given thread ID.

If the target *target_thread* ID is **0**, **pthread_join( )** will return with error **ESRCH**.

EDEADLK A deadlock was detected or the value of *target_thread* specifies the calling thread. (See NOTES section below.)

SEE ALSO

wait(2), **pthread_create**(3T), **pthread_exit**(3T), **pthread_join**(3T)

NOTES

Using **thr_join**(3T) in the following syntax,

while (thr_join(NULL, NULL, NULL) == 0);

will wait for the termination of all other undetached and non-daemon threads; after which, **EDEADLK** will be returned.

pthread_join(3T), on the other hand, must specify the *target_thread* ID for whose termination it will wait.

Calling **pthread_join( )** also "detaches" the thread, that is, **pthread_join( )** includes the effect of **pthread_detach( )**. Hence, if a thread were to be cancelled when blocked in **pthread_join( )**, an explicit detach would have to be done in the cancellation cleanup handler. In fact, the routine **pthread_detach( )** exists mainly for this reason.