pthread_create - create a new thread

SYNOPSIS

#include <pthread.h>

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine) (void *), void *arg);

DESCRIPTION

The new thread terminates in one of the following ways:

* It calls pthread_exit(3), specifying an exit status value that is available to another thread in the same process that calls pthread_join(3).
* It returns from start_routine(). This is equivalent to calling pthread_exit(3) with the value supplied in the return statement.
* It is canceled (see pthread_cancel(3)).
* Any of the threads in the process calls exit(3), or the main thread performs a return from main(). This causes the termination of all threads in the process.

The attr argument points to a pthread_attr_t structure whose contents are used at thread creation time to determine attributes for the new thread; this structure is initialized using pthread_attr_init(3) and related functions. If attr is NULL, then the thread is created with default attributes.

Before returning, a successful call to pthread_create() stores the ID of the new thread in the buffer pointed to by thread; this identifier

is used to refer to the thread in subsequent calls to other pthreads functions.

RETURN VALUE

On success, pthread_create() returns 0; on error, it returns an error number, and the contents of *thread are undefined.

pthread_join - join with a terminated thread

SYNOPSIS

#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);

DESCRIPTION

The pthread_join() function waits for the thread specified by thread to terminate. If that thread has already terminated, then pthread_join() returns immediately. The thread specified by thread must be joinable.

If retval is not NULL, then pthread_join() copies the exit status of the target thread (i.e., the value that the target thread supplied to pthread_exit(3)) into the location pointed to by retval. If the target thread was canceled, then PTHREAD_CANCELED is placed in the location pointed to by retval.

If multiple threads simultaneously try to join with the same thread, the results are undefined. If the thread calling pthread_join() is canceled, then the target thread will remain joinable (i.e., it will not be detached).

RETURN VALUE

On success, pthread_join() returns 0; on error, it returns an error number.

pthread_exit - terminate calling thread

SYNOPSIS

```
#include <pthread.h>
void pthread_exit(void *retval);
```

DESCRIPTION

The pthread_exit() function terminates the calling thread and returns a value via retval that (if the thread is joinable) is available to another thread in the same process that calls pthread_join(3).

Any clean-up handlers established by pthread_cleanup_push(3) that have not yet been popped, are popped (in the reverse of the order in which they were pushed) and executed. If the thread has any thread-specific data, then, after the clean-up handlers have been executed, the corresponding destructor functions are called, in an unspecified order.

When a thread terminates, process-shared resources (e.g., mutexes, condition variables, semaphores, and file descriptors) are not released, and functions registered using atexit(3) are not called.

After the last thread in a process terminates, the process terminates as by calling exit(3) with an exit status of zero; thus, process-shared resources are released and functions registered using atexit(3) are called.

RETURN VALUE

This function does not return to the caller.

pthread_cancel - send a cancellation request to a thread

SYNOPSIS

#include <pthread.h>

int pthread_cancel(pthread_t thread);

DESCRIPTION

The pthread_cancel() function sends a cancellation request to the thread thread.

A thread's cancelability state, determined by pthread_setcancel-state(3), can be enabled (the default for new threads) or disabled. If a thread has disabled cancellation, then a cancellation request remains queued until the thread enables cancellation. If a thread has enabled cancellation, then its cancelability type determines when cancellation occurs.

A thread's cancellation type, determined by pthread_setcanceltype(3), may be either asynchronous or deferred (the default for new threads). Asynchronous cancelability means that the thread can be canceled at any time (usually immediately, but the system does not guarantee this). Deferred cancelability means that cancellation will be delayed until the thread next calls a function that is a cancellation point. A list of functions that are or may be cancellation points is provided in pthreads(7).

When a cancellation requested is acted on, the following steps occur for thread (in this order):

1. Cancellation clean-up handlers are popped (in the reverse of the order in which they were pushed) and called. (See pthread_cleanup_push(3).)
2. Thread-specific data destructors are called, in an unspecified order. (See pthread_key_create(3).)
3. The thread is terminated. (See pthread_exit(3).)

The above steps happen asynchronously with respect to the pthread_cancel() call; the return status of pthread_cancel() merely informs the caller whether the cancellation request was successfully queued.

After a canceled thread has terminated, a join with that thread using pthread_join(3) obtains PTHREAD_CANCELED as the thread's exit status. (Joining with a thread is the only way to know that cancellation has completed.)

RETURN VALUE

On success, pthread_cancel() returns 0; on error, it returns a nonzero error number.