Cancelling a Thread

What Happens When You Cancel a Thread?

When you cancel a thread, the cancellation request is sent to the target thread. The target thread can then respond to this request based on its current cancelability state and type. There are a few key points to consider:

1. **Cancelability State:**
    - **PTHREAD_CANCEL_ENABLE:** The thread can be canceled.
    - **PTHREAD_CANCEL_DISABLE:** The thread cannot be canceled.
2. **Cancelability Type:**
    - **PTHREAD_CANCEL_DEFERRED:** The default type where the thread will only respond to cancellation requests at defined cancellation points (e.g., `pthread_testcancel()`, blocking functions like `sleep()`, `read()`, etc.).
    - **PTHREAD_CANCEL_ASYNCHRONOUS:** The thread can be canceled immediately, but this can be unsafe and is generally discouraged.

When a thread is canceled, it performs cleanup routines by invoking cleanup handlers set by `pthread_cleanup_push()`and `pthread_cleanup_pop()` before termination.

How to Cancel a Thread

You can cancel a thread using the `pthread_cancel` function.

Syntax

c
Copy code

```c
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

Example

c
Copy code

```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *thread_function(void *arg) {
    printf("Thread started\n");
    for (int i = 0; i < 10; ++i) {
        printf("Thread running: %d\n", i);
        sleep(1);
    }
    return NULL;
}

int main() {
    pthread_t thread;

    // Create a thread
    pthread_create(&thread, NULL, thread_function, NULL);

    // Sleep for 3 seconds to let the thread run
    sleep(3);

    // Cancel the thread
    pthread_cancel(thread);

    // Wait for the thread to finish
    pthread_join(thread, NULL);

    printf("Main thread finished\n");
    return 0;
}
```

Why Cancel a Thread?

Thread cancellation is useful in scenarios where:

- A long-running or potentially blocking thread needs to be stopped based on some condition.
- You need to gracefully shut down a multi-threaded application.
- You want to stop a thread that is no longer needed.

Changing the Cancel State

Changing the cancel state of a thread can be done using pthread_setcancelstate and pthread_setcanceltype.

Changing Cancel State

Use pthread_setcancelstate to enable or disable cancellation.

c
Copy code
```c
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>

void *thread_function(void *arg) {
    int oldstate;

    // Disable cancelability
    pthread_setcancelstate(PTHREAD_CANCEL_DISABLE, &oldstate);
    printf("Thread running in non-cancelable state\n");
    sleep(5); // Simulate long processing

    // Restore previous cancelability state
    pthread_setcancelstate(oldstate, NULL);
    printf("Thread running in cancelable state\n");
    sleep(5); // Simulate more processing

    return NULL;
}

int main() {
    pthread_t thread;

    // Create a thread
    pthread_create(&thread, NULL, thread_function, NULL);

    // Sleep for a short period to ensure the thread runs
    sleep(1);

    // Try to cancel the thread
    pthread_cancel(thread);

    // Wait for the thread to finish
    pthread_join(thread, NULL);

    printf("Main thread finished\n");
    return 0;
}
```

Changing Cancel Type

Use pthread_setcanceltype to set the cancellation type to deferred or asynchronous.

c
Copy code
```c
#include <pthread.h>

int pthread_setcanceltype(int type, int *oldtype);
```

- type: Can be PTHREAD_CANCEL_DEFERRED or PTHREAD_CANCEL_ASYNCHRONOUS.
- oldtype: Pointer to store the old cancellation type.

Example

c
Copy code
```c
#include <pthread.h>
#include <stdio.h>
```

```c
#include <unistd.h>

void *thread_function(void *arg) {
    int oldtype;

    // Set asynchronous cancelability
    pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, &oldtype);
    printf("Thread running in asynchronous cancelable state\n");
    sleep(5); // Simulate long processing

    // Restore previous cancelability type
    pthread_setcanceltype(oldtype, NULL);
    printf("Thread running in default cancelable state\n");
    sleep(5); // Simulate more processing

    return NULL;
}

int main() {
    pthread_t thread;

    // Create a thread
    pthread_create(&thread, NULL, thread_function, NULL);

    // Sleep for a short period to ensure the thread runs
    sleep(1);

    // Try to cancel the thread
    pthread_cancel(thread);

    // Wait for the thread to finish
    pthread_join(thread, NULL);

    printf("Main thread finished\n");
    return 0;
}
```

In practice, deferred cancellation is preferred because it is safer and allows the thread to clean up resources properly. Asynchronous cancellation can be unpredictable and is generally discouraged unless absolutely necessary.