Linux Threads: Pthread API

Kishan Kumar Ganguly
Asst. Prof.
IIT, DU

Pthread API Overview

- Defines core functionality for multithreaded programs
- Over 100 interfaces
- Widely used on Unix systems
- API Defined in <pthread.h>
- Function Prefix: pthread_
    - Example: pthread_create() for thread creation

Pthread Function Groups

- **Thread Management**
  - Create, destroy, join, and detach threads
- **Synchronization**
  - Manage thread synchronization
  - Mutexes, condition variables, and barriers

Linking Pthreads

- Separate library: **libpthread**
- Requires explicit linkage
- Use -pthread flag with gcc
- Ensures proper library linkage and thread safety controls

```
gcc -Wall -Werror -pthread my_program.c -o my_program
```

Introduction to Creating Threads

- Transitioning to Multithreading
- Initial program execution is single-threaded
- Creating additional threads for multithreading
- **Key function:** pthread_create()

pthread_create() Function

- **Function Signature:**

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

- Creates a new thread and executes start_routine
- Arguments explained: thread, attr, start_routine, arg

start_routine Signature

- Function executed by the new thread
- Signature: void *start_thread(void *arg)
- Takes a void pointer as argument and returns a void pointer
- Similar to fork() but threads share resources, not copies

Error Handling

- Handling pthread_create() Errors
- Possible error codes: EAGAIN, EINVAL, EPERM

```
pthread_t thread;

int ret;

ret = pthread_create(&thread, NULL, start_routine, NULL);

if (ret != 0) {

    errno = ret;

    perror("pthread_create");

    return -1;

}

    /* A new thread is created and running start_routine concurrently ... */
```

Thread IDs

- Understanding Thread IDs (TID)
- Equivalent to Process IDs (PID) for threads
- Assigned by the Pthread library
- Represented as an opaque type, pthread_t

Obtaining Thread IDs

- Obtaining the TID of a Thread
- TID provided in pthread_create()
- Runtime access with pthread_self() function
- Simple and error-free usage:

```
#include <pthread.h>

pthread_t pthread_self(void);

const pthread_t me = pthread_self();
```

Comparing Thread IDs

- Thread IDs (TID) vs. Process IDs (PID)
- Pthread standard doesn't guarantee pthread_t to be an arithmetic type
- Special interface for comparing thread IDs: pthread_equal()
- Using pthread_equal()
- Returns nonzero if thread IDs are equal, 0 if not
- Simple example:

```
int ret;

ret = pthread_equal(thing1, thing2);

if (ret != 0)

    printf("The TIDs are equal!\n");

else

    printf("The TIDs are unequal!\n");
```

Terminating Threads

- Thread Termination vs. Process Termination
- Threads can terminate due to:
    - Returning from start routine
    - Invoking pthread_exit()
    - Being canceled by another thread

Termination Methods

- Terminating the Calling Thread
- Use pthread_exit() to terminate the calling thread
- No chance of error, similar to exit()

```
pthread_exit(NULL);
```

Terminating Other Threads

- Terminating Threads from Another Thread
- Use pthread_cancel() for thread cancellation
- Async or deferred cancellation types
- Example of enabling cancellation and sending a cancellation request:

```
int unused;
int ret;
ret = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, &unused);
if (ret) {
    errno = ret;
    perror("pthread_setcancelstate");
    return -1;
}
```

```
ret = pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, &unused);
if (ret) {
    errno = ret;
    perror("pthread_setcanceltype");
    return -1;
}


/* `thread` is the thread ID of the to-terminate thread */
ret = pthread_cancel(thread);
if (ret) {
    errno = ret;
    perror("pthread_cancel");
    return -1;
}
```

Joining and Detaching Threads

- Synchronizing Threads with Joining
- **pthread_join()** function allows one thread to block until another terminates
- **pthread_join()** Function
- Blocks the invoking thread until the specified thread terminates
- Returns the terminated thread's return value if retval is not NULL
- Allows threads to synchronize their execution
- **Possible error codes:** EDEADLK, EINVAL, ESRCH

Joining Threads Example

```
int ret;

/* Join with `thread' and ignore its return value */

ret = pthread_join(thread, NULL);

if (ret) {

    errno = ret;

    perror("pthread_join");

    return -1;

}
```

Detaching Threads

- Detaching Threads
- Threads are created as joinable by default
- Use **pthread_detach()** to make a thread non-joinable
- Frees system resources when the thread terminates
- pthread_detach() Function
    - Detaches the specified thread
    - Returns zero on success, ESRCH if the thread is invalid
- Recommended to call either pthread_join() or pthread_detach() for each thread to release resources

Detaching vs. Joining Threads - Example

```c
int ret;
pthread_t thread_to_join, thread_to_detach;
/* Create a thread to join */
ret = pthread_create(&thread_to_join, NULL, start_routine, NULL);
if (ret) {
    errno = ret;
    perror("pthread_create");
    return -1;
}
/* Create a thread to detach */
ret = pthread_create(&thread_to_detach, NULL, start_routine,
NULL);
if (ret) {
    errno = ret;
    perror("pthread_create");
    return -1;
}
```

```c
/* Joining the first thread */
ret = pthread_join(thread_to_join, NULL);
if (ret) {
    errno = ret;
    perror("pthread_join");
    return -1;
}

/* Detaching the second thread */
ret = pthread_detach(thread_to_detach);
if (ret) {
    errno = ret;
    perror("pthread_detach");
    return -1;
}
```

Classwork

- How to create and manage multiple threads in a C program using **pthreads** for concurrent execution?

Classwork

https://snipit.io/public/lists/24735/82648

Thank You!