

Threads, IPC, Synchronization in Linux

Chapter 12

Beginning Linux Programming

Chapter 12

THREADS

Thread

- All the program that we have written so far have had **just one** thread of execution
- Time to do many tasks within a single program
- We need many threads of execution in a single program
- **Writing multithreaded programs requires very careful design**
- In Linux we would use POSIX Threads, usually referred to as Pthreads.

Thread

- The subroutines which comprise the Pthreads API can be informally grouped into
- three major classes:
 - **Thread management**
 - **Mutexes**
 - **Semaphores**

Threads

`pthread_create` creates a new thread, much as `fork` creates a new process.

```
#include <pthread.h>
```

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

```
#include <pthread.h>
```

```
void pthread_exit(void *retval);
```

```
#include <pthread.h>
```

```
int pthread_join(pthread_t th, void **thread_return);
```

```
$ cc -D_REENTRANT thread1.c -o thread1 -lpthread
```

Thread Creation

`pthread_create` creates a new thread, much as `fork` creates a new process.

```
#include <pthread.h>
```

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

- Arguments
 - pointer to pthread object (used as thread **identifier**)
 - the thread attributes (NULL for us)
 - the address of a **function** taking a pointer to void as a parameter and the function will return a pointer to void
 - the thread would execute this
 - we can pass any type of single argument and return a pointer to any type (needs **typecasting**)
 - argument passed to this function
- Returns 0 on success

Simple Example

```
#include<stdio.h>
#include<pthread.h>
#include<stdlib.h>

void * threadFunc1(void * arg){
    int i;
    for(i=1;i<=5;i++){
        printf("%s\n", (char*)arg);
        sleep(1);
    }
}

int main(void){
    pthread_t thread1;
    pthread_t thread2;
    char * message1 = "I am thread 1";
    char * message2 = "I am thread 2";

    pthread_create(&thread1,NULL,threadFunc1,(void*)message1 );
    pthread_create(&thread2,NULL,threadFunc1,(void*)message2 );
    while(1);
    return 0;
}
```

Thread Exit

- When a thread terminates, it calls the `pthread_exit` function
 - terminates the calling thread,
 - returns a pointer to an object

```
#include <pthread.h>
```

```
void pthread_exit(void *retval);
```


Thread Join

- The first parameter is the thread for which to wait
 - the identifier that `pthread_create` filled in for us.
- The second argument is a pointer to a pointer that itself points to the return value from the thread.

```
#include <pthread.h>
```

```
int pthread_join(pthread_t th, void **thread_return);
```

Thread Join

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

void *thread_function(void *arg);

char message[] = "Hello World";

int main() {
    int res;
    pthread_t a_thread;
    void *thread_result;

    res = pthread_create(&a_thread, NULL, thread_function, (void *)message);
    if (res != 0) {
        perror("Thread creation failed");
        exit(EXIT_FAILURE);
    }
    printf("Waiting for thread to finish...\n");
    res = pthread_join(a_thread, &thread_result);
    if (res != 0) {
        perror("Thread join failed");
        exit(EXIT_FAILURE);
    }
    printf("Thread joined, it returned %s\n", (char *)thread_result);
}
```

Thread Join

```
    printf("Message is now %s\n", message);  
    exit(EXIT_SUCCESS);  
}  
  
void *thread_function(void *arg) {  
    printf("thread_function is running. Argument was %s\n", (char *)arg);  
    sleep(3);  
    strcpy(message, "Bye!");  
    pthread_exit("Thank you for the CPU time");  
}
```

Requirements

- we must **define** the **macro** `_REENTRANT`
- **include** the file `pthread.h`, and
- **link** with the threads **library** using `-lpthread` or `-pthread`
- To compile & link a source file named `thread`
 - `gcc -D_REENTRANT thread.c -o thread -lpthread`
- Run
 - `./thread`

Chapter 14

SEMAPHORES

Semaphores in Thread

```
#include <semaphore.h>
```

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
```

```
#include <semaphore.h>
```

```
int sem_wait(sem_t * sem);
```

```
int sem_post(sem_t * sem);
```

```
#include <semaphore.h>
```

```
int sem_destroy(sem_t * sem);
```

Mutex

```
#include <pthread.h>
```

```
int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t  
*mutexattr);
```

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```