ITF22519: Introduction to Operating Systems

Fall Semester, 2021

Lab6: Thread Programming 1

Submission Deadline: October 8th, 2021 23:59

In this lab, you will learn how to create threads, terminate threads, wait for a thread, and write programs that use threads. Before you start, remember to `commit` and `push` your previous lab to your git repository. Then, try to `pull` the new lab:

```
$ cd Introduction2OS/labs
$ git pull upstream main
$ cd lab6
```

From this lab assignment, all your code in `.c` files and the output of your code have to follow the requirements of each specific exercise unless nothing else stated.

1 Thread Creation and Termination

To create a thread in a C program, the following statements are needed:

```
#include <pthread.h>
...
pthread_t Thread;
...
int ret = pthread_create(&Thread, NULL, (void *)startRoutine, NULL);
...
```

These lines would create a variable named `Thread` which will hold the ID of a newly created thread, courtesy of the function call `pthread_create()` and will be used to perform various functions on the thread in subsequent pthread calls.

The function call `pthread_create()` takes in as arguments:

- a buffer to a `pthread_t` variable (`&Thread` in this case)
- a pointer to a thread attribute structure (the first NULL)
- a function pointer to a function that the `Thread` is to perform (`void *(startRoutine)` here)
- a pointer to the arguments to the function that the `Thread` is to perform (the final NULL in the example)

To terminate a thread in a C program, use `pthread_exit(NULL)`.

As an example, the following piece of code that creates and terminates 10 threads each of which prints out a hello message:

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>

#define NUM_OF_THREADS     10

void *PrintMessage(void *ThreadId){
       long tid;
       tid = (long)ThreadId;
       printf("Hello World from Thread #%ld!\n", tid);
       pthread_exit(NULL);
}

int main (int argc, char *argv[]){
      pthread_t threads[NUM_OF_THREADS];
      int ret;
      long i;

      for(i=0; i<NUM_OF_THREADS; i++){
            printf("Creating Thread %ld in the main() function\n", i);
            ret = pthread_create(&threads[i], NULL, PrintMessage, (void *)i);
            if (ret){
                 printf("ERROR in creating thread; return ERROR code %d\n", ret);
                 exit(-1);
            }
      }
      pthread_exit(NULL);
      return 0;
}
```

Save this code into a file called *pthread_create.c*, and to compile the code, run gcc on the source file, but with a few extra arguments as follows:

```
gcc pthread_test.c -pthread -o pthread_test
```

Note: Be aware the `-pthread` flag. This is needed to ensure that the pthread library is linked during the linking phase of gcc execution.

Task 1: Run the program produced by compiling *pthread_create.c* and answer the following questions:.

- What is expected output of the program?
- Explain the output.

2 Waiting a Thread

As explained in the man page for `pthread_create()`, a thread that is created would terminate if the main thread returns from main, even when the created thread has not completed its task yet. To ensure that the main thread waits until the created threads complete before it continues, the function call `pthread_join()` is used. An example of how to use it is as follows:

```
#include <pthread.h>
...
pthread_t Thread;
...
int ret = pthread_create(&Thread, NULL, (void *)startRoutine, NULL);

pthread_join(Thread, NULL);
...
```

The above example will ensure that `Thread` is joined to the main thread, and the main thread will not terminate before `Thread` has finished execution of startRoutine.

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>
#include <unistd.h>

#define NUM_OF_THREADS 10

void *PrintMessage(void *ThreadId){
       long tid;
       tid = (long)ThreadId;
       printf("Hello World from Thread #%ld!\n", tid);
       sleep (2);
}

int main (int argc, char *argv[]){
      pthread_t threads[NUM_OF_THREADS];
      int ret;
      long i;

      for(i=0; i<NUM_OF_THREADS; i++){
            printf("Creating Thread %ld in the main() function\n", i);
            ret = pthread_create(&threads[i], NULL,PrintMessage, (void *)i);
            if (ret){
                 printf("ERROR in creating thread; return ERROR code %d\n", ret);
                 exit(-1);
            }
            pthread_join(threads[i], NULL);
     }
     return 0;
}
```

Task 2: Run the program produced by the above code and answer the following questions:

- Explain the output.
- How the output is different from that of **Task 1**.

3 Exercises

3.1 Exercise 1 (50 pts)

A C program can have several threads each of which can do different tasks. The program in *Ex1.c* file aims to print out the following output but it is not completed.

```
Initial values: a = 0, b = 1000
Increasing b finished thanks to the main thread!
Increasing a finished thanks to the second thread!
Final values: a= 100, b = 100
```

Tasks on this exercise:

- Complete the code in *Ex1.c* file to have the expected output.
- Save your code in a *Exercise1.c* file.

Note: You are recommended to increase/decrease the values of a and b step by step.

3.2 Exercise 2 (50 pts)

In the multi-thread C program in *Ex2.c* file, each thread is expected to print out its message before the main function prints out its own message. However, one output of the program is as follows:

```
This is Thread 1
I am the main thread.
```

- Explain why the message from Thread 2 is missing.
- Fix the bug in the program so that each Thread completes its tasks. Add the output in your report.
- Save your code into a new *Exercise2.c* file.

3.3 Exercise 3 (20 pts)

In the input square matrices `A.dat` and `B.dat`, the first element indicates the size of each matrix. Here, the size of each matrix is 64x64. Write a C program that performs matrix multiplication for these two matrices using `pthreads`. `A.dat` and `B.dat` are the first and the second matrix, respectively. The specification for the program is as follows:

- Using 8 threads each of which is responsible for calculating 8 rows of the output matrix.
- If any errors are encountered during program execution, the program prints out the error information and then exits.
- The output file is the same format as the input files.

Note: This lab assignment is not mandatory. However, students who perform **Exercise 3** will get bonus points on the final total scores.

4 What To Submit

Complete the exercises in this lab. Then, put all of files into the **lab6** directory of your repository. Make a report for each exercise. After that, run `git add .` and `git status` to ensure the file has been added and commit the changes by running `git commit -m "Commit Message"`. Finally, submit your files to GitHub by running `git push`. Check the GitHub website to make sure all files have been submitted.