LE/EECS 3221 – Operating System Fundamentals

Summer 2020

Programming Assignment 2

Submission Deadline: July 12, 2020 before 23:59

Objectives

In this assignment we will try to practice the concept of multithreaded processes with Pthread library.

General Assignment Notes

When writing and submitting your assignments follow these requirements:

- Name your source code file as: your YorkU student number, an underscore, 'a' (for 'assignment', then the assignment number in two digits. For example, if the user 100131001 submits Assignment 1, the name should be: `100131001_a01.c.txt`. No other file name format will be accepted. We require the .txt extension in the end because Moodle does not allow .c extension.
- Use the same naming scheme for the assignment title when submitting the assignment to the Moodle.
- For this assignment you must use C99 language syntax. Your code must compile using make **without errors and warnings.** You will be provided with a makefile and instructions on how to use it. If you use that makefile to compile the code, then you don't have to do anything special to select C99.
- **Test your program thoroughly with the `gcc` compiler (version 5.4.0) in a Linux environment.** This can be verified by running the command "gcc --version" in Linux shell.
- If your code does not compile, **then you will get zero**. Therefore, make sure that you have removed all syntax errors from your code.

Marks will be deducted from any question(s) where these requirements are not met.

WARNING

Follow the assignment instructions to the letter in terms of the file names and function names, as this assignment will be auto graded. If anything is not as per description, the auto grading will fail, and your assignment will be given a mark of 0.

Synopsis

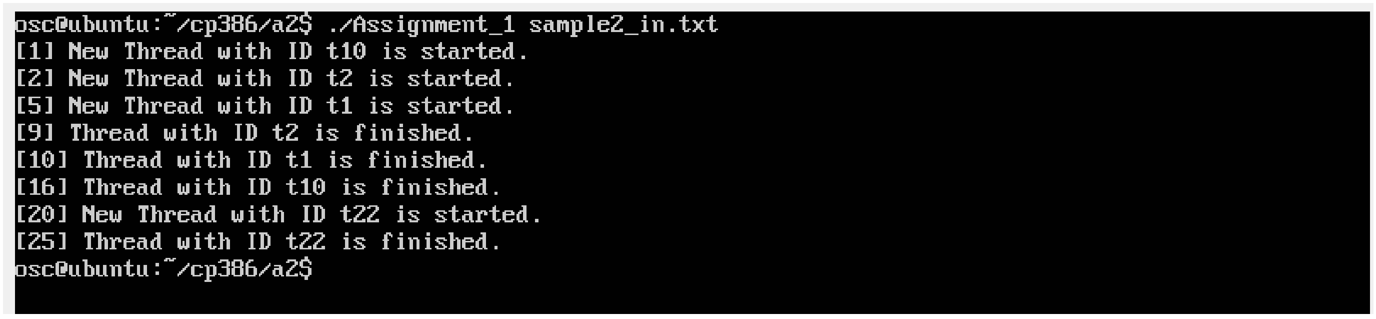
In this assignment, you will have a process that will create multiple threads at different times. These threads may have different start_time and lifetime. Once all the threads are over, the process will terminate. There are three actions that must be performed in this process:

- Read the input file that contains thread properties: ID, start time and lifetime.
- Create a thread whenever it is the start time of some thread.
- All the threads implement/run same function. Implement this function so that the thread should terminate when it has consumed time units equal to its lifetime.

Description

For this assignment, you are provided a skeleton code in the file `student_code.c`. Some functions are completely implemented, some are partially and for some only header is provided. Additionally, you can write your own functions if required. Complete the program as per the following details so that we can have functionality as described in the Synopsis above. Write all the code in single C file:

1. The program has its local clock mechanism. You must invoke `startClock()` when you are ready to service threads. The file reading should be performed before this. Once the `startClock()` is invoked, the `getCurrentTime()` can be invoked to see how much time units are passed since `startClock()` was invoked. This will provide a local clock ticks behavior to service your threads.
2. The thread information will be provided to this program in a text file. You must accept the name of this file as command line argument. In this file, there will be one thread information per line. Each thread information has three attributes provided in the order: `thread_id;start_time;lifetime`. It is not necessary that the individual thread lines are in any specific order. `start_time` indicates that at what time a thread should be created and started. `lifetime` is the time units for which a thread must stay alive. Use the methods in step 1 in a suitable way to implement this functionality.
3. To read the file, `readFile()` function is provided. However, it has a small part missing that you must complete so that the thread attributes can be stored in memory in a suitable way. This also requires the completion of `struct thread`. Add suitable members to `struct thread` and store the content of the file in a suitable data structure.
4. Once you have thread information in memory, start servicing the threads using the suitable POSIX library calls. `main()` is provided but you have to complete it in a suitable way. The program must keep running if there are some threads which are not yet created or if there are some running threads.
5. The `threadRun()` is the function that each thread must run. Implement this function so that each thread takes care of its `lifetime` and terminates when that is over.
6. To compile the program, use the provided makefile.
7. The image below shows the expected output for the sample input file provided with this assignment:



```
osc@ubuntu:~/cp386/a2$ ./Assignment_1 sample2_in.txt
[1] New Thread with ID t10 is started.
[2] New Thread with ID t2 is started.
[5] New Thread with ID t1 is started.
[9] Thread with ID t2 is finished.
[10] Thread with ID t1 is finished.
[16] Thread with ID t10 is finished.
[20] New Thread with ID t22 is started.
[25] Thread with ID t22 is finished.
osc@ubuntu:~/cp386/a2$
```