LE/EECS 3221 – Operating System Fundamentals

Summer 2020

Programming Assignment 3

Submission Deadline: August 2, 2020 before 23:59

Objectives

In this assignment we will try to practice the concept of synchronization with semaphores, deadlocks and starvation. The assignment extends the concepts of multithreading used in A02.

Permitted similarity threshold for this assignment is 90%.

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

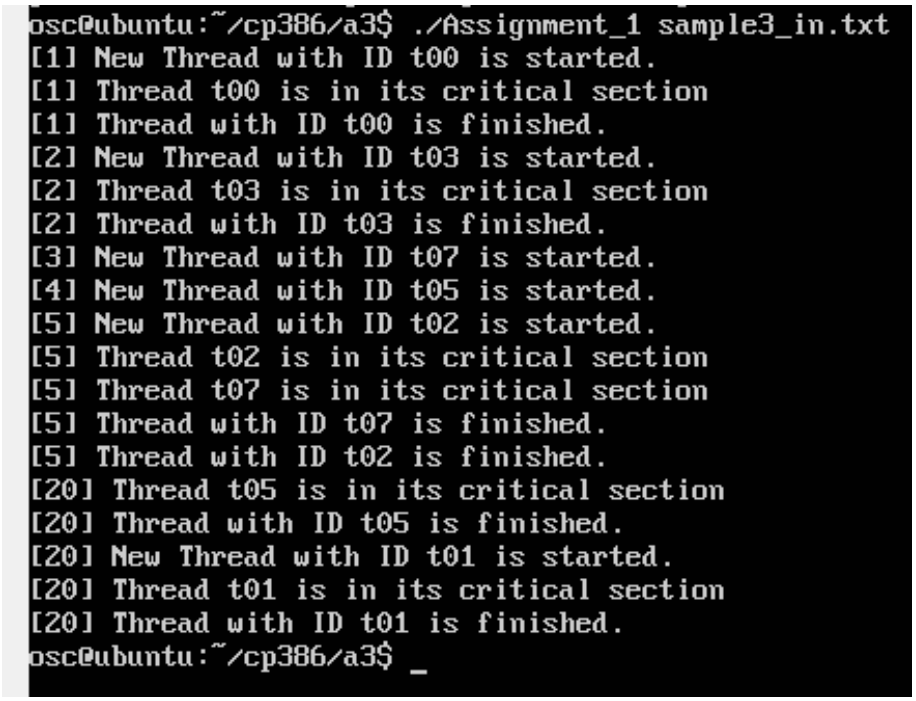
In this assignment, our process will create multiple threads at different times, as in A02. These threads may have different `start_time` but there is no lifetime. Each thread after its creation runs a small critical section and then terminates. All threads perform same action/code. Most of the code such as reading the input file, creating the threads etc. is provided. Your task is to implement following synchronization logic with the help of POSIX semaphores:

- Only one thread can be in its critical section at any time in this process.
- The first thread, in terms of creation time, enters first in its critical section.
- After that threads are permitted to perform their critical section based on their ID.
    - Threads are given IDs in the format `txy` where `x` and `y` are digits (0-9). Thread IDs are unique. Threads may have same or different start_time. Thread entries in the input file can be in any order.
    - The "`y`" in thread IDs thus will either be an even digit or odd digit.
    - After the first thread, the next thread that will be permitted to perform its critical section must be the one in which "`y`" is different i.e. if "`y`" was even in first thread then in the next it must be odd or vice versa.
    - For the rest of the process, you must follow the same scheme i.e. two threads with odd "`y`" or even "`y`" can not perform critical section simultaneously.
- Since synchronization may lead to deadlock or starvation, you have to make sure that your solution is deadlock free i.e. your program must terminate successfully, and all the threads must perform their critical section.
- One extended form of starvation will be that towards the end, we have all odd or all even processes left, and they are all locked because there are no other type (odd/even) of threads left. Once the process reaches to that stage, you must let them perform their critical section to avoid starvation and progress towards the end of the process. **However, you must make sure that there are no other threads coming in future which could help avoid this situation. If there is chance for more threads coming, then you will hold this till then.**

Description

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

1. The code provided reads the content of file for you and populate the threads information in a dynamic array of type `struct thread`. You may add some more members to this data structure if required. If you want to initialize those members, then you can possibly do that during the file read.
2. The `main()` already contains the code to create and invoke threads. However, there is no synchronization logic added to it. If required, you will add some suitable code in the while loops to perform the tasks required.
3. The `threadRun()` function also contains the code that a thread must run. However, again the synchronization logic is missing. Add the suitable code before and after the critical section.
4. You will need to create and use POSIX semaphore(s) to implement the required logic.
5. The image below shows the expected output for the sample input file provided with this assignment:


osc@ubuntu:~/cp386/a3$ ./Assignment_1 sample3_in.txt
[1] New Thread with ID t00 is started.
[1] Thread t00 is in its critical section
[1] Thread with ID t00 is finished.
[2] New Thread with ID t03 is started.
[2] Thread t03 is in its critical section
[2] Thread with ID t03 is finished.
[3] New Thread with ID t07 is started.
[4] New Thread with ID t05 is started.
[5] New Thread with ID t02 is started.
[5] Thread t02 is in its critical section
[5] Thread t07 is in its critical section
[5] Thread with ID t07 is finished.
[5] Thread with ID t02 is finished.
[20] Thread t05 is in its critical section
[20] Thread with ID t05 is finished.
[20] New Thread with ID t01 is started.
[20] Thread t01 is in its critical section
[20] Thread with ID t01 is finished.
osc@ubuntu:~/cp386/a3$ _