LE/EECS 3221 – Operating System Fundamentals

Fall 2022

Programming Assignment 2

Submission Deadline: November 20, 2020 before 23:59

Objectives

In this assignment we will try to practice the concepts such as: multithreading, synchronization with semaphores, deadlocks and starvation.

Permitted similarity threshold for this assignment is 80%.

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 2, the name should be: `100131001_a02.c.txt`. No other file name format will be accepted. We require the .txt extension in the end because eCourse does not allow .c extension.
- **Use the same name the assignment title when submitting the assignment to the eClass.**
- Compile your assignment using "make", do not directly compile using gcc.
- **Test your program thoroughly in a Linux environment.**
- If your code does not compile, **then you will get zero**. Therefore, make sure that you have removed all syntax errors from your code.
- Do not leave any testing printing statements in 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. 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 pthreads and 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. In the screen shot on the next page, you will notice that t07, t05 and t01 perform their critical section without any even number thread separating them because there are no more even number threads left. **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. For example, in the screenshot on the next page, this started happening at t=20 which is the creation time of the last thread in our input file.**

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. **DO NOT forget to rename your C code file appropriately when submitting**:

1. The code provided reads the content of file for you and populate the threads information in a dynamic array of type `struct thread`. This list of threads is populated for you in the `threads`. Also, you have `threadCount` variable available to see the number of threads that are populated from the input file. You may add some more members to this data structure if required. If you want to initialize those additional members, then you can possibly do that during the file read.

2. The `main()` already contains the code to read threads from input file. However, there is no code to initialize, execute and synchronize threads. You have to perform these tasks in a suitable way there. `startClock()` invocation as given in `main()` is required to initiate the program's internal clock, so do not remove it.
3. The `threadRun()` function also contains the code that a thread must run. However, the synchronization logic (entry/exit section) is missing. Add the suitable code before and after the critical section.
4. You will need to create and use POSIX pthreads and semaphore(s) to implement the required logic.
5. The image below shows the expected output for the sample input file (sample2_in.txt) provided with this assignment. In this output when there are multiple threads finishing (or may be starting) at the same time, e.g. at t=5 both t02 and t07 are finished, then their order may switch and may be different than their critical section order, because start/finish is out of critical section and unsynchronized. However, the critical section order, e.g. in t=5 both t02 and t07 perform their critical section, must always be as per our synchronization requirement. Also, you have to make sure that a thread must be started at its creation time as per the input file i.e. "is started" message of thread must have the same time stamp as mentioned in the input file: