CS/SE 4348 Operating Systems
Project 3: Due date 26th Mar, 2018

This project can be done individually or with a partner. Sharing your work with other groups is strictly prohibited.

Seeking Tutor Problem

The computer science department runs a mentoring center to help undergraduate students with their programming assignments. The lab has a coordinator and several tutors to assist the students. The waiting area of the center has several chairs. Initially, all the chairs are empty, the coordinator is waiting for students to arrive and all the tutors are waiting for students to be assigned by coordinator. A student who arrives at the center seeking help sits in an empty chair in the waiting area and waits to be called for tutoring. If no chairs are available, the student will come back at a later time. If the coordinator sees a student waiting, then the coordinator wakes up an idle tutor, if there is one, and assigns the student to the tutor. If all the tutors are busy assisting students, the coordinator waits for a tutor to get free and then assigns a waiting student to the tutor. A tutor after assisting a student, waits for the next student to be assigned to him

Using POSIX threads, mutex locks, and semaphores, implement a solution that coordinates the activities of the coordinator, tutors, and the students. Some hints to implement this project are provided next.

Implementation Hints

Using Pthreads (Section 4.4.1), begin by creating n students and m tutors as separate threads. (n and m are arguments to the program.) The coordinator will run as a separate thread. Student threads will alternate between programming for a period of time and seeking help from the tutor. If the tutor is available, they will obtain help. Otherwise, they will either sit in a chair in the waiting area or, if no chairs are available, will resume programming and seek help at a later time.

When a student arrives and finds an empty chair, the student must notify the coordinator using a semaphore. When the tutor is free (initially and after helping a student), the tutor must notify the coordinator using a semaphore. Also, waiting students and tutors must be woken up by the coordinator using separate semaphores. For details on how to use pthreads synchronization primitives mutex and semaphore see section 5.9.4 of the book. For a detailed tutorial on Pthreads and Semaphores, see https://computing.llnl.gov/tutorials/pthreads/#Thread

To simulate students programming in students threads, and the tutor providing help to a student in the tutor thread, the appropriate threads should sleep (by invoking sleep()) for a random of time (up to three seconds).

The total number of students, the number of tutors, the number of chairs, and the number of times a student seeks a tutor's help are passed as command line arguments as shown below (csmc is the name of the executable):

```
csmc #students #tutors #chairs #help
csmc 10 3 4 5
```

Once a student thread takes the required number of help from the tutors, it should terminate. Once all the student threads are terminated, the tutor threads, the coordinator thread, and finally the main program should be terminated.

Output

Your program must output the following at the appropriate times:

Student [#] takes a seat. Waiting students = [# of students waiting]
Student [#] found no empty chair will try again later
Tutor [#] helping student for [#] seconds. Waiting students = [# of students waiting]

Grading Policy

Name your program file as csmc.c/cpp and submit it through elearning.

Implementation: 50%. Source code should be structured well with adequate comments clearly showing the different parts and functionalities implemented.

Correctness: 50%.

The TA will compile the code and test it in cs1.utdallas.edu. So, your program should run correctly in cs1.utdallas.edu. Add a note at the top of the source code in case you are using any special flags for compilers/linkers.

Search the Web!!

This project is a modification of sleeping barber (TA) problem. There is a plethora of sites in which the solution for the problem is available. Feel free to use these sources to understand the problem and the solution if you will. But, you are not allowed to use the code that is available in the public domain. Also, before searching the web think of the solution by yourself first.

Programming concurrency is fun. Enjoy!!!