**CS 370 – Project #3**

Purpose: Become familiar with multi-threaded programming, semaphores, and deadlock concepts
Points: 150 Scoring will include functionality, documentation, and coding style

**Introduction:**


Consider a University professor who helps students with their programs during office hours. The professor's office is rather small and has room for only one desk with a chair and computer. There are three chairs in the hallway outside the office where students can sit and wait if the professor is currently helping another student. When there are no students who need help during office hours, the professor takes a nap. If a student arrives during office hours and finds the professor sleeping, the student must awaken the professor to ask for help. If a student arrives and finds the professor currently helping another student, the student sits on one of the chairs in the hallway and waits. If no chairs are available, the student will leave and come back at a later time.

From the description of the problem, we will use semaphores[1] and implement two threads;

1. **`studentsThd()`** - which can either get tutored or will leave if there is no free chair
2. **`professorThd()`** - which can either tutor students or go to sleep

In this problem, the professor represents the operating system and the students represent processes. The operating system should allocate the required resources to the processes and, at the same time, avoid deadlock. There are many various too this problem, so be sure to follow the provided guidance. Solutions downloaded from the internet will be referred to the Office of Student Conduct.

***Note***, scoring will include functionality, comments, and general code quality.

**Project:**

Implement the Sleeping professor problem in **C** (not C++). The program must compile and execute under Ubuntu 18.04 LTS (and will only be testing with Ubuntu). Assignments not executing under Ubuntu will not be scored. See additional guidance on following pages.

**Submission**

When complete, submit:

- A copy of the **C** source code (not C++) file by class time.

Submissions received after the due date/time will not be accepted.

1 For more information, refer to: https://en.wikipedia.org/wiki/Semaphore_(programming)

**Specifications**

The ***main()*** function should perform the following actions:

- Display header
- Read (scanf) and validate student count
  - Must be between STUDENT_COUNT_MIN and STUDENT_COUNT_MAX (inclusive)
  - Refer to example execution
- Student Help Counts
  - Create and populate an array for the number of times each student will need help.
    - initialize each student help count (i.e., `(rand()%HELPS_MAX + 1);`)
    - the array must be dynamically allocated (i.e., use `malloc()`).
- Threads array (students)
  - An array for student threads must be used (one per student)
    - Thread array must be dynamically allocated (i.e., use `malloc()`).
- Initialize semaphores
  - The student semaphore initialized to 0
  - The professor semaphore initialized to 0
  - The chairs semaphore array all initialized to 0
- Create one professor thread
- Create *student count* student threads
- Wait for the appropriate threads to complete.

The ***professor()*** function will perform the following actions

- Loop forever
  - Wait until awakened by a student
  - After awakened, display message
  - Once awakened, loop to help all waiting students
    - Student vacates a chair and enters office
      - if chair count is 0, exit help students loop
      - decrement chair count (must use mutex)
      - student vacates chair (applicable chair semaphore)
        - display message (see example)
      - professor helps the student (random amount of time)
        - display message (see example)
        - random amount of time: `usleep(rand() % 1500000);`
      - signal that next student can enter
  - If all students have been helped (all helps counts 0).
    - exit thread

The ***students()*** function should perform the following actions.

- Loop forever
  - Display message that student is working on the assignment (see example)
  - Student works on the assignment for a random amount of time
    - Random amount of time: `usleep(rand() % 2000000);`
  - Student decides to ask for help (display message)
  - Get chair count (must use mutex)
  - If no chairs available
    - Display message, continue working on the assignment
  - If a chair is available
    - If all chairs available

        - awaken the sleeping professor
    - Occupy a chair
        - update chair count (must use mutex)
        - wait until professor is available to help (applicable chair semaphore)
        - when professor helps, display message
        - wait until professor is done helping
        - decrement the help count for that student
        - if that students help count is 0
            - exit thread

The professor and student messages should be printed in red and the student messages printed in green.

```
printf("\033[0;31mI'm a message in red\033[0m\n", 0);     // red
printf("\033[0;92mI'm a message in green\033[0m\n", 0);   // green
printf("\033[0;94mI'm a message in blue\033[0m\n", 0);    // blue
printf("\033[0;93mI'm a message in yellow\033[0m\n", 0);  // yellow
printf("\033[0;1mI'm an underlined message\033[0m\n", 0);
```

Refer to the example execution for examples of the coloring.

**Include Files**

In order to use threading functions and semaphores in C, you will need the below include files:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <semaphore.h>
#include <time.h>
#include <stdbool.h>
```

Use the following parameters

```
#define   STUDENT_COUNT_MIN   2
#define   STUDENT_COUNT_MAX   10
#define   CHAIR_COUNT         3
#define   HELPS_MAX           3
```

The semaphores, mutexes, and some integers will be declared globally.

**Compilation Options**

Use the following compiler options

```
gcc -Wall -pedantic -pthread -o sleepingProf sleepingProf.c
```

Points will be removed for poor coding practices and unresolved warning messages.

## Example Execution

```
ed-vm% ./sleepingProf
CS 370 - Sleeping Professor Project

How many students coming to professor's office? 6

Student 1 doing assignment.
Student 2 doing assignment.
Student 3 doing assignment.
Student 4 doing assignment.
Student 5 doing assignment.
Student 6 doing assignment.
Student 1 needs help from the professor.
Student 2 needs help from the professor.
Professor has been awakened by a student.
Student frees chair and enters professors office. Remaining chairs 2
Professor is helping a student.
Student 1 is getting help from the professor.
Student 3 needs help from the professor.
Student 5 needs help from the professor.
Student 4 needs help from the professor.
Chairs occupied, student 4 will return later.
Student 4 doing assignment.
Student 6 needs help from the professor.
Chairs occupied, student 6 will return later.
Student 6 doing assignment.
Student 1 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 2 is getting help from the professor.
Student 4 needs help from the professor.
Student 6 needs help from the professor.
Chairs occupied, student 6 will return later.
Student 6 doing assignment.
Student 2 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 3 is getting help from the professor.
Student 6 needs help from the professor.
Student 1 needs help from the professor.
Chairs occupied, student 1 will return later.
Student 1 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 5 is getting help from the professor.
Student 1 needs help from the professor.
Student 2 needs help from the professor.
Chairs occupied, student 2 will return later.
Student 2 doing assignment.
Student 5 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 4 is getting help from the professor.
```

```
Student 2 needs help from the professor.
Student 4 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 6 is getting help from the professor.
Student 5 needs help from the professor.
Student 4 needs help from the professor.
Chairs occupied, student 4 will return later.
Student 4 doing assignment.
Student 6 doing assignment.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 1 is getting help from the professor.
Student 4 needs help from the professor.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 2 is getting help from the professor.
Student 6 needs help from the professor.
Student frees chair and enters professors office. Remaining chairs 1
Professor is helping a student.
Student 5 is getting help from the professor.
Student 5 doing assignment.
Student frees chair and enters professors office. Remaining chairs 2
Professor is helping a student.
Student 4 is getting help from the professor.
Student frees chair and enters professors office. Remaining chairs 3
Professor is helping a student.
Student 6 is getting help from the professor.
Student 5 needs help from the professor.
Student frees chair and enters professors office. Remaining chairs 3
Professor is helping a student.
Student 5 is getting help from the professor.

All students assisted, professor is leaving.
ed-vm%
```

*Note*, actual mileage may vary.