Programming Assignment 2: Concurrency
Due: October 17th, 2018 5:30PM

Description

In this assignment you will gain hands-on experience with parallel programming and the difficulty of building correct parallel programs. You are tasked with helping your professor schedule his office hours. The professor is teaching 2 classes this semester, class A and class B, and is holding shared office hours for both classes in his office. The professor can have a large number of students showing up for his office hours, so he decides to impose several restrictions.

Functional Requirements

Requirement 1: The professors's office has only 3 seats, so no more than 3 students are allowed to simultaneously enter the professor's office. When the office is full and new students arrive they have to wait outside the office.

Requirement 2: The professor gets confused when helping students from class A and class B at the same time. He decides that while students from class A are in his office, no students from class B are allowed to enter, and the other way around.

Requirement 3: The professor gets tired after answering too many questions. He decides that after helping 10 students he needs to take a break before he can help more students. So after the 10th student (counting since the last break) enters the professors office no more students are admitted into the office, until after the professors's next break. Students that arrive while the professor is taking his break have to wait outside the office.

Requirement 4: In order to be fair to both classes after 5 consecutive students from a single class the professor will answer questions from a student from the other class.

Requirement 5: Your program should ensure progress, i.e. if there is no student in the professor's office and the professor is not currently taking a break an arriving student should not be forced to wait. Similarly, if an arriving student is compatible with the students currently in the office he should not be forced to wait, unless the professor is due for a break.

Requirement 6: Your code shall not deadlock.

Non-Functional Requirements

Requirement 7: Your source file shall be named officehours.c. The source file must be ASCII text files. No binary submissions will be accepted.

Requirement 8: Tabs or spaces shall be used to indent the code. Your code must use one or the other. All indentation must be consistent.

Requirement 9: No line of code shall exceed 100 characters.

Requirement 10: Each source code file shall have the following header filled out:

```
/*

  Name: Student Name
  ID:   10000001

*/
```

Requirement 11: All code must be well commented. This means descriptive comments that tell the intent of the code, not just what the code is executing.

The following are poor comments.

```
// Set working_str equal to strdup return
char *working_str  = strdup( cmd_str );

// Set working_root equal to working_str
char *working_root = working_str;
```

The following explains the intent:

```
// Save a copy of the command line since strsep
// will end up moving the pointer head
char *working_str  = strdup( cmd_str );

// we are going to move the working_str pointer so
// keep track of its original value so we can deallocate
// the correct amount at the end
char *working_root = working_str;
```

When in doubt over comment your code.

Requirement 11: Keep your curly brace placement consistent. If you place curly braces on a new line , always place curly braces on a new end. Don't mix end line brace placement with new line brace placement.

Requirement 12: Each function should have a header that describes its name, any parameters expected, any return values, as well as a description of what the function does. For example

Requirement 13: Remove all extraneous debug output before submission. The only output shall be the output of the commands entered or the shell prompt.

Requirement 14: Your code shall compile cleanly on omega.uta.edu with no warnings using: `gcc officehours.c -o office hours -lpthread`

Administrative

You have been provided a framework for the simulation, which creates a thread for the professor and a thread for each student who wants to attend the office hour. You will need to add synchronization to ensure the above restrictions.

The source code for this assignment can be found on the course website at:

`https://github.com/CSE3320/Office-Hours-Assignment`

The student threads are implemented in the functions classa_student() and classb_student(), which simulate students from class A and class B, respectively. After being created a student from class A executes three functions: he enters the office (classa_enter()), he asks questions (ask_questions()) and then leaves the office (classa_leave()). A student from class B calls the corresponding functions classb_enter(), ask_questions() and classb_leave(). All synchronization between threads should be added in the ..._enter() and ..._leave() functions of the students and the function professorthread(), which implements the professor.

The simulation framework is implemented in the file officehours.c. The program expects as an argument the name of an input file which controls the simulation. The input file specifies the arrival of students and the amount of time students spend in the professor's office. More precisely, the file has one line for each student containing two numbers. The first number is the time (in seconds) between the arrival of this student and the previous

student. The second number is the number of seconds the students needs to spend with the professor.

The provided code implements all the functionality for the students and the professor, but does not implement any synchronization. To help you in developing your code a number of assert statements that help you check for correctness have been added. DO NOT delete those assert statements. Also, do not make any changes to the functions ask_questions and take_break. You might want to add additional assert statements, for example for ensuring that the number of students since the last break is less than the limit, or for ensuring that there are not class A and class B students in the office at the same time.

Before you start your work, compile officehours.c and try to run it on this sample input file sample_input.txt. This will simulate 3 different students asking questions for 25, 10 and 15 seconds, respectively. The time between the first and the second student arriving is 10 seconds and the time between the second and the last student is 5 seconds. .

This assignment must be coded in C. Any other language will result in 0 points. You programs will be compiled and graded on `omega.uta.edu`. Please make sure they compile and run on `omega` before submitting them. Code that does not compile on omega with:

```
gcc officehours.c -o office hours -lpthread
```

will result in a 0.

Your program, officehours`.c` is to be turned in via blackboard. Submission time is determined by the blackboard system time. You may submit your programs as often as you wish. Only your last submission will be graded.

You may use no other outside code.

Academic Integrity

This assignment must be 100% your own work. No code may be copied from friends, previous students, books, web pages, etc. All code submitted is automatically checked against a database of previous semester's graded assignments, current student's code and common web sources. By submitting your code on blackboard you are attesting that you have neither given nor received unauthorized assistance on this work. **Code that is copied from an external source or used as inspiration, excluding the course github or blackboard, will result in a 0 for the assignment and referral to the Office of Student Conduct.**