Coursework 1 – the project marking problem

Introduction

The problem described here is a slightly anarchic variant of the system that was formerly used by the Department to assess MSc projects (with 2 markers per student). These days, the markers attend separate presentations.

There are S students on a course. Each student does a project, which is assessed by K markers. There is a panel of M markers, each of whom is required to assess up to N projects. We assume that S*K ≤ M*N. Note that there is no guarantee that all students will complete their demos during the session even with this constraint.

Near the end of the course a session is arranged at which each student demonstrates their project to K markers (together). The session lasts T minutes and each demonstration takes exactly D minutes. Students enter the lab at random intervals during the session except in the last D minutes. All markers are on duty at the beginning of the session and each remains there until they have attended N demos or the session ends. At the end of the session all students and markers must leave the lab. Moreover, any students and markers who are not actively involved in a demo D minutes before the end must leave at that time, because there would be no time to complete a demo that started later.

Whenever a student enters the lab, they start by finding K idle markers (one at a time). If there are any markers available, they will be "grabbed" by the student. Having grabbed K markers, the student does the demo and then leaves the lab. Each marker stays idle until grabbed by a student, waits until other idle markers are found (during which time the marker cannot be grabbed by other students), attends the demo and then becomes idle again, leaving the lab after attending N demos.

Deliverables

This coursework is worth 50% of the marks on the Server Software (COMSM2001) unit.

You must submit the following on SAFE:

1. The source code for your program in .c and .h files. You may place these in a single .zip file if you want – but you shouldn't need too many files.
2. A makefile called **`makefile`** that produces an executable named **`demo`** when the command **`make`** is run on one of the lab machines.
3. A report in a single text file called **`report.txt`** with at most 5000 characters.

The most important criterion for your program is correctness. After that, we will consider efficiency and simplicity.

We recommend that you use the c99 standard for your code.

Please read the "instructions and marking" section for more information.

The problem

The aim of the exercise is to write a C program that simulates the demo session described above. Each student and each marker must be represented by a separate thread, using the Posix threads standard. A skeleton program is provided for you.

A student thread should execute the following steps:

1. Panic for a random time between 0 and (T-D-1) minutes..
2. Enter the lab.
3. Grab K markers when they are idle.
4. Do a demo for D minutes.
5. Exit from the lab.

A marker thread executes the following steps:

1. Enter the lab.
2. Repeat (N times):
    a. Wait to be grabbed by a student.
    b. Wait for the student's demo to begin.
    c. Wait for the student's demo to finish.
3. Exit from the lab.

However, if the end of the lab session approaches and there is not time for another demo left then all students and markers not currently in a demo immediately exit the lab. In this case, the corresponding threads print a “timeout” message.

All parts not directly related to threading are implemented for you already in the skeleton program. For example, the function panic() is step 1. of the student thread and step 4. is just a call to demo().

Since your program will be run with some automated tests, do not modify the printf statements in the skeleton program. You can move them around if you change the structure of the program.

The skeleton program runs the demo with time scaled up so that 1 minute becomes 10ms. If you want to pause a certain number of “minutes”, call `delay(int t)` where t is the number of minutes. For example, the demo() function is simply delay(D) as a demo takes D minutes.

You must not busy-wait under any circumstances.

Your program should work correctly for all reasonable values of S, M, K, N, T, and D (that is, meeting the constraints checked by the main function). Your program should work correctly for any scheduling strategy; i.e., regardless of the relative speeds of the threads.

The only parts of the program that you need to modify are the marker() and student() threads and possibly the run() function. In addition, you may want to add global variables for synchronization.

Instructions and marking

This coursework is worth 50% of the marks on the Server Software (COMSM2001) unit.

The following will be taken into account, among other things, when marking your submission. Anything not meeting these rules will incur a penalty.

- You must submit both a report and the source code of your program. Failing to submit both gets you 0 marks.
- You must not submit the compiled program.
- Your program must compile to an executable named "demo" and run when typing "make" followed by "demo" on a lab machine. It is your own responsibility to test before you submit. If your program does not compile or run due to basic C syntax/programming errors you may get 0 marks.
- All files that you submit must be your own work. Sharing code with other students or copying code from the internet are examples of plagiarism.
- If you use any material in your submission that you did not create yourself then you must reference it properly.
- Your program will be tested automatically and then marked by hand. For this reason it is important that your program's output is exactly in the specified format.
- You must use Posix threads.
- You must not use busy-waiting.

You must use error handling for any Posix thread function that can fail. Failing to do so will incur a penalty. The basic rules:

- If a function returns an int which indicates a possible error, always store the return value and then check it before you proceed any further.
- In a multithreaded program, error handling must be thread-safe too. Don't use a global "int err" variable – it has to be local to the thread.
- In a situation where an error really shouldn't happen, call exit(1), abort() or something like that. If this ever occurs then you know you have a bug. You can use a macro or helper function to print diagnostic information if you like.

Your report must be a text file (not word, PDF or anything else) and the limit is 5000 characters. It must contain three parts:

1. How you solved the problem of synchronizing your threads.
2. Why your method is correct (or how it could be made correct).
3. Why your method is efficient (or how it could be improved).

Do not include any program trace output in your report. Instead, you need to give a convincing argument in English why your program is correct and efficient, or how it could be improved.