COMP 304: Project 2
t Seconds of Fame

[redacted]

Notes: The project can be done **individually or in teams of 2**. You may discuss the problems with other teams [redacted] but the submitted work must be your own work. This assignment is worth 12% of your total grade.

[redacted]

Introduction

In this project, you will get more familiar with the concepts of scheduling, synchronisation, multi-threading and deadlock prevention in operating systems by using POSIX Threads (pthreads) API.

Background

A media company would like to air some new debate programs on TV. However, they want to test the waters first, since they are not sure how their format would work. That's why they asked an OS student to make a simulator for the program.

The program features several commentators and a single moderator. Since this is supposed to be a civil TV program, only one person can speak at a given time. This should be ensured by the moderator.

More importantly, the speakers should not get into a deadlock, since a program will not be very interesting if it features people sitting quietly.

Your task is to make a simulator which satisfies these conditions.

Part 1 (70 pts)

The program format is based on a Q&A model. The moderator asks a question to start. Each commentator requests to answer to this question with probability p. After some of the commentators give their opinions on the subject, a new question is asked. There will be a total of q questions.

You decide to model the commentators and the moderator as separate threads. In order to answer

a question, a commentator thread will post a request to a global queue. Afterwards, it waits for a "your turn to speak" signal from the moderator and "speaks" (read: sleeps) for t_{speak} seconds.

When the commentator is done speaking, they should notify the moderator. If the queue is empty and no one is interested in speaking, the moderator asks a new question and the cycle continues.

Here are more implementation details for Part 1:

- All threads are created at program start.
- There should be N commentator threads and one moderator thread. Specify N with a command line option `-n`.
- The moderator asks q questions in total. (specified by `-q` option)
- Asking a question does not require screen time.
- After a question is asked, each commentator thread requests to answer with probability p (specified by `-p` option).
- If a request to answer is generated, the commentator thread posts a request to the global queue and waits for a signal.
- The moderator pops the first request in the queue and wakes up the associated commentator thread.
- When the commentator thread wakes up, it should speak (sleep) for t_{speak} seconds and notify the moderator.
- t_{speak} should be **a random real number** between 1 and t (specified by `-t` option).
- Only one commentator should speak at a time.
- If the queue is empty and no commentator is speaking, the moderator asks a new question.
- The program ends when all questions are asked and all requests to answer have been processed.
- Join all threads before program completion.

Part 2 (20 pts)

Sometimes a breaking news event occurs during the program, such as a fire, an earthquake or an important person giving a speech. The channel owners would like you to adapt to this scenario.

Such events should be generated by the main thread. Each second, if there isn't a breaking event, the main thread should generate a breaking event with probability b. Specify b with command line option `-b`.

You should have a separate thread which waits for breaking news events. This thread will log the start and the end of the breaking news event. The event always lasts 5 seconds.

In addition, when the breaking event occurs, any ”speaking” commentator thread should cut short and the moderator should stop processing requests until the event ends. The program will continue with the next speaker after the break.

Logging (10 pts)

You should print logs to stdout for demonstration and debugging purposes. Suggested logging points are when:

- the moderator asks a question
- a commentator creates a request to speak
- a commentator starts and finishes speaking
- a breaking event starts and ends

Add timestamps **with milliseconds** to your logs. You can utilize the `gettimeofday` function for that purpose.

The suggested format is:

```
...
[00:04.195] Moderator asks question 2
[00:04.195] Commentator #1 generates answer, position in queue: 0
[00:04.195] Commentator #0 generates answer, position in queue: 1
[00:04.195] Commentator #1's turn to speak for 1.561 seconds
[00:05.756] Commentator #1 finished speaking
[00:05.756] Commentator #0's turn to speak for 2.958 seconds
[00:07.001] Breaking news!
[00:07.001] Commentator #0 is cut short due to a breaking news
[00:12.001] Breaking news ends
[00:12.001] Moderator asks question 3
...
```

Deliverables

Submit a zip file with:

- your source (.c, .cpp, .h) files
- your Makefile
- a README briefly describing your implementation and which parts work
- Logs of a sample run with parameters: `-n 4 -p 0.75 -q 5 -t 3 -b 0.05`. This simulation should take about 30-40 seconds.

Randomly selected projects will be invited to perform a demo on 7-8th of June.

Tips

- Start simple: first implement a program with fixed parameters and a single commentator.
- Don't forget to lock the global queue.
- Adding a `-seed` command line argument to seed the RNG will help a lot with debugging.
- Use the `pthread_sleep` function provided on Blackboard to sleep pthreads. Don't use `sleep()`.
- `pthread_cond_timedwait` might help with the breaking news implementation.
- Attend the PS and office hours to get more tips.

Good luck!