COMP 304: Project 2
Electronic Voting System

Due: May, 11th, 2023

Notes: The project must be done with a partner or individually. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own team. This assignment is worth (14)% of your total grade.

Corresponding TAs: Mohammad Issa (missa18)

Electronic Voting Simulation

The Republic of Banana is planning to upgrade its voting process by introducing electronic voting stations for the upcoming presidential election. However, the election committee intends to test the new electronic voting system with a simulator before implementing it in the actual elections.

Your task in this project is to develop a simulator that can replicate an election day scenario. The simulator should include the modeling of independent voters waiting in queues to cast their votes, polling stations for voting, and regular maintenance in case of any failures. It is crucial to ensure that the simulation is race-free and deadlock-free to prevent any miscounts or disruptions during the voting process.

In this project, you are required to use the POSIX threads (pthreads) API to implement the simulation. You will get more familiar with the concepts of scheduling, synchronisation, multi-threading and deadlock prevention in operating systems.

Simulation Settings

Each voter will be voting at a single polling station and will cast their vote to one for the presidential candidate. The presidential candidates are Mary, John, and Anna, with 40%, 15%, and 45% chances of getting elected, respectively.

To simulate the election day scenario, there are four parts that need to be implemented, each building upon the previous one in increasing complexity. It is recommended to implement the simulation parts in order, starting from Part 1 and moving towards Part 4. This way, you can build upon each previous part's functionality and ensure that the entire simulation is working correctly.

Part I

(40 points) When a voter arrives at the polling station, they will need to click on the queue machine to get their position number in the queue. A separate screen will display the next voter (signals/wakes up) who should go to the polling station. Once their turn comes, the voter will go to the polling station and cast their votes for the presidential candidate of their choice. Here are more detailed rules for the simulation environment.

- The simulation should use real-time. Get the current time of the day, run the simulation until current time + simulation time.
- There is a single polling station that is used.
- There is only one voter using the polling station at any time.
- A new voter arrives to the polling station with probability p at every t secs and uses the queuing machine to get their position on the queue.
- Each voter takes $2t$ secs to cast their vote (sleep the thread for $2t$ secs).
- The vote can be to any of the aforementioned candidates, with the following probabilities (Mary: 40, John: 15, Anna: 45)
- After a voter finishes voting, they have no consequence on the simulation.
- It is not allowed for a voter to use the polling station without their queue number being shown on the display (a voter cannot acquire a lock from another voter to access the polling station).
- Use command line arguments *-t* and *-p* to indicate the total simulation time (e.g. -t 200) and probability (e.g. 0.5), respectively.

In real life, t is usually in the order of minutes but assume t is 1 sec for the purpose of the computer simulation.

Part II

(30 points) In this section, we will add a priority system for certain voters. The queuing machine favors the elderly and pregnant women voters (assume that the voter enters their national identity number to get a queuing position). Now in addition to creating a voter with probability p at every t, create a pregnant/elderly voter with probability *1-p* at every t. The queuing machine must favor elderly or pregnant women and let them vote until one of following conditions hold:

- (a) No more elderly or pregnant women are waiting,
- (b) 5 or more non-elderly non-pregnant voters are lined up to vote.

Note that this solution now causes starvation to the elderly and pregnant voters. Suggest and implement a solution to avoid starvation of the elderly and pregnant voters. Briefly explain why starvation for the elderly and pregnant voters occurs and how you solve it in your report. You can play with p to enforce starvation to test your solution.

Part III

(10 points) Now, we will simulate failure for the polling station. Assume that every $10t$ secs, the polling station fails with probability f and the voters have to wait until the polling station is fixed by a mechanic (who has the highest priority). Fixing the station takes $5t$ secs, once fixed, the queuing machine will signal the next voter to use the polling. Of course, while the station is being fixed, the new voters may arrive as usual. Again, you can change with the probability to enforce failure to test your solution. Make f as a command line argument.

Part IV

(20 points) Now instead of having a single polling station, there will be multiple polling stations. Each polling station will have its own queuing machine. Make sure the voting counters for each polling station are independent. Whenever a new voter arrives, assign him/her to the polling station which is least busy. Busyness here is evaluated based on the number of people waiting to cast their votes. Each of the new polling stations can still fail the same way noted in part III, and the polling machine's respective voters will have to wait. Make *-c* as a command line argument to specify the number of polling stations in the simulation.

Keeping Logs

(10 points) You are required to keep a log for the voters and for the queuing machine, which will help you debug and test your code. The voters.log should keep the station ID, voters ID, their category (special (S) or ordinary (O) or mechanic (M)), the request time the voters requested their queue position, the polling station time (end of polling station usage), turnaround time (polling station time - request time).

StationID.VoterID	Category	Request Time	Polling Station Time	Turnaround Time
1.1	S	0	2	2
1.2	O	0	4	4

Output the snapshot of the waiting voters with their IDs and categories, in addition to the current votes, in every second starting from n^{th} secs to the terminal, where n is also a command line argument. The numbers indicates the voter IDs waiting in the queue at time n. Feel free to come up with a better representation or GUI.

Example output:

```
At 20 sec, polling station 1, elderly/pregnant: 16, 18, 20
At 20 sec, polling station 1, ordinary      : 23, 25
At 20 sec, polling station 2, elderly/pregnant: 17, 19
At 20 sec, polling station 2, ordinary      : 21, 24, 26
At 20 sec total votes: (Mary: 4, John: 2, Anna: 3)
...
```

Also to initialize the queuing simulation, make sure the following is done:

- At time zero, there is one none-elderly none-pregnant woman voter waiting to vote and one elderly or pregnant woman waiting to vote. (in total 2 waiting)

Implementation

- You may want to keep a queue for high priority voters and another queue for other types of voters, add voters to queues at the appropriate times. C++ STL queues may help your implementation of the data structure.
- You can use random number generator to generate voters at the specified probability. For easy debugging, add a command line argument for a seed (-s #) and feed the seed to the random number generator. (Very important for debugging).
- You should represent each voter as a thread. The queuing machine should have its own separate thread.
- Start simple. For example simulate Part 1 and make sure it works correctly. Add complexity as you move to the other parts.
- To sleep pthreads, please use the code that we provided on blackboard. Do not use sleep() system call.
- For Pthread semaphores, mutexes and condition variables, also refer to the pthread tutorial online: https://hpc-tutorials.llnl.gov/posix/

Deliverables

You are required to submit the followings packed in a zip file (named your-username(s).zip) to blackboard :

- .c or .cpp source file that implements the simulation. Please comment your implementation.
- sample log files for 60 sec simulation for p=0.5.

- any supplementary files for your implementation (e.g. Makefile)
- a README (report) file describing your implementation, particularly which parts of your code work. Since we cannot hold a demo with everyone, document your report properly.
- We will share the github classroom link soon.
- Finally you may perform a demo if requested by the TA if there are issues with your implementation.

GOOD LUCK.