COMP 304: Project 2
Traffic Simulation with Pthreads

Due: Friday May 3rd, midnight

Notes: The project can be done **individually or teams of 2**. You may discuss the problems with other teams and post questions to the OS discussion forum but the submitted work must be your own work. This assignment is worth 10% of your total grade.

Corresponding TA for the project : Aditya Sasongko msasongko17@ku.edu.tr

Description

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

Traffic Intersection Simulation

In this project, you will implement multiple independently acting vehicles in traffic by modeling a busy intersection, where the traffic is controlled by a traffic police officer (say due to the power outage). Your job is to implement the simulation so that no deadlock or traffic accident occurs.

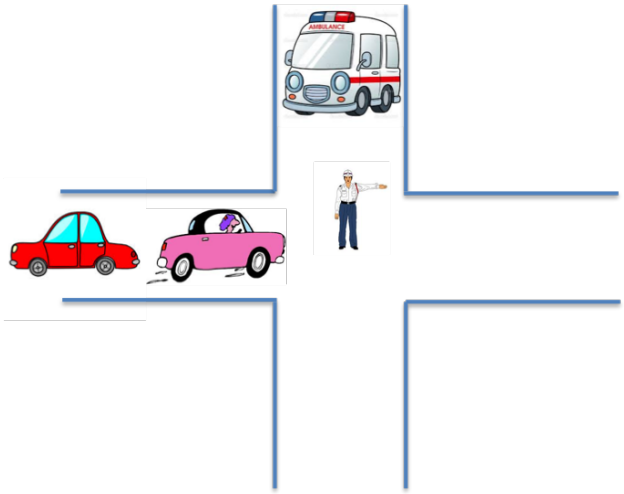

Figure 1: Traffic directed by a police officer due to the blackout

Part I

(50 points) Here are the rules for the simulation environment:

- The simulation should use real-time. Get the current time of the day in sec and run the simulation until current time + simulation time.
- This is a four-way intersection where cars can come from all four directions (North, South, West, and East (N, S, W and E)).
- Traffic only flows to one direction at a time. That is, if cars coming from N are crossing the intersection, the cars on all other directions have to wait.
- A car can turn any of the four directions and has no consequence on the simulation.
- Each car takes 1 second to go through the intersection (sleep the thread for one sec).
- A car arrives to S with probability p at every second, (similarly with probability p to W and to E) but to North with probability *1-p*. Make p a command-line argument.
- From North, once no car comes, there is a 20 second delay (sleep) before any new car will definitely arrive. Then cars will start arriving again with *1-p*.
- For efficiency, the police officer lets the current lane to flow until one of following conditions hold
  - (a) No more cars on this lane,
  - (b) 5 cars or more are lined up at any of the other lanes
- The police officer gives priority to the lane that has the highest number of cars lined up to cross the intersection. In case of a tie, the directions have the following priority order: $N > E > S > W$ (N has the top priority).
- At time zero, there is one car in each direction.
- Add a command line argument -*s* to indicate the total simulation time (e.g. -s 200).
- Finally, assume that the officer has four eyes, two in the back, and he has no hearing problems.

Part II

(20 points) Part I may cause starvation to the lanes that have a lower priority resulting a driver wait forever to cross the intersection. In this section, we will add a maximum wait-time for the drivers to avoid this situation.
The police officer now lets the current lane to flow until one of following conditions hold

- (a) No more cars on this lane,
- (b) 5 cars or more are lined up at any of the other lanes or

- (c) Any car on the other lanes has been waiting for 20 secs or more.

For the police officer's safety, if both (b) and (c) hold true, he will first give way to (c) so that he is not beaten up by an angry driver due to the starvation.

Part III

(20 points)
If no car is around, the traffic officer starts playing games on his cell phone. When a car arrives at the intersection, it will honk the horn to notify the officer, which will result in 3 sec delay for the officer to react and navigate the traffic. You should use a condition variable to notify the police officer.

Keeping Logs

(10 points)
You are required to keep a log for the cars and for the police officer, which will help you debug and test your code. The car.log should keep the car no (ID), its incoming direction, the arrival time, the crossing time, wait-time (crossing time - arrival time). The police.log should keep the events for cell phone usage and honk times.

CarID	Direction	Arrival-Time	Cross-Time	Wait-Time
1	N	18:19:10	18:19:11	1
2	S	18:19:10	18:19:12	2

Time	Event
18:21:23	Cell Phone
18:21:37	Honk

Output the snapshot of the intersection every t, $t+1$, $t+2$ secs to the terminal, where t is also a command line argument. The numbers indicate how many cars there are in each lane at time t. Here are sample outputs:
At 18:20:10:

```
    5
4       0
    1
```

At 18:20:11:

```
    4
4       1
    2
```

At 18:20:12:

```
    3
4       1
    2
```

Suggestions

- You may want to keep a queue for each lane, add cars 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 cars at the specified probability. For easy debugging, add a command line argument for a seed and feed the seed to the random number generator.
- You should represent each lane as a thread. The police officer should have its own separate thread.
- Start simple. For example, simulate two lanes first (W and E). Add other lanes and complexity as you are sure your implementation works (no deadlock, no accident). Implement Part III last.
- 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: http://www.llnl.gov/computing/tutorials/pthreads/

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=0.4.
- any supplementary files for your implementation (e.g. Makefile)
- a README file briefly describing your implementation, particularly which parts of your code work.

GOOD LUCK.