COMP 304: Project 2
Metro Simulation

Due: Sunday, April 29th, 11.59 pm

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 9% of your total grade. START EARLY. **Any material you use from web should be properly cited in your report. Any sort of cheating will be harshly PUNISHED.**

Corresponding TA for the project : Burak Bastem

Description

This project allows you to get more familiar with the concepts of scheduling, synchronisation, multi-threading and deadlock prevention in operating systems by using POSIX threads (pthreads) API.

Metro Simulation

You will simulate a Metro tunnel and a Metro control center in a metropolis during a rush hour. The tunnel interconnects four different sections and has only one lane due to geographical conditions. Each section has two lanes outside of the tunnel for each direction, one for incoming trains and one for outgoing trains. The scenario is illustrated in Figure 1. Your job is to implement the simulation so that no deadlock or train accident occurs.

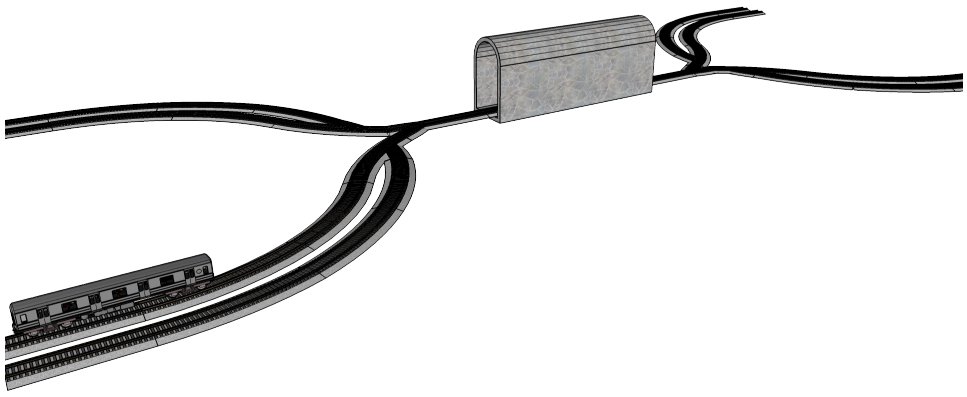

Figure 1: Metro tunnel interconnecting four different sections, each section has two lanes allowing traffic on both directions

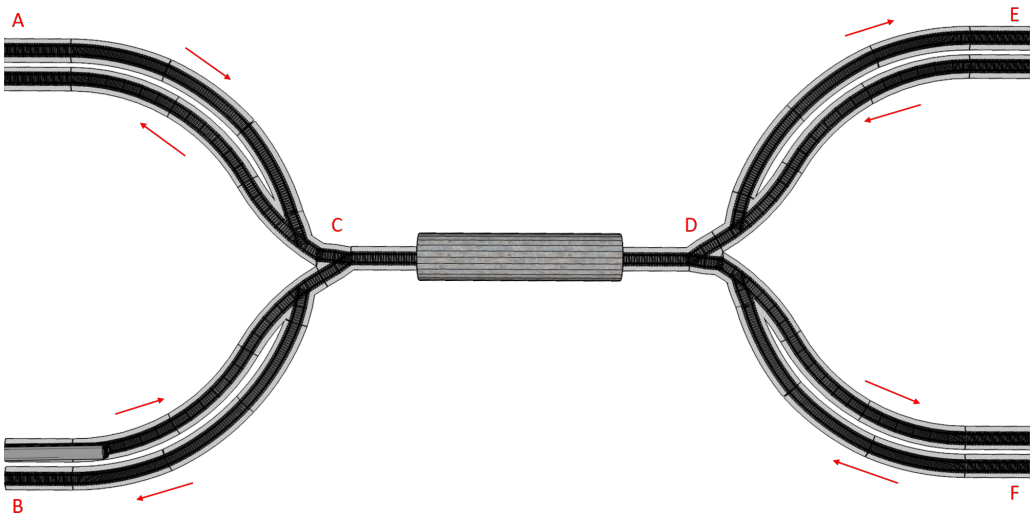

Figure 2: Labels for sections

Part I

(50 points) Here are the rules for the simulation environment (use Figure 2 as a reference):

- The simulation should use real-time. Get the current time of the day, and run the simulation until current time + simulation time. The simulation time is in secs.
- There are four different Metro lines:
  - Line-1: between A and E
  - Line-2: between A and F
  - Line-3: between B and E
  - Line-4: between B and F
- The section containing tunnel (C-D) has only one lane. That is if there is a train passing through this section, other trains have to wait.
- Each train travels A-C, B-C, D-E or D-F sections in 1 second regardless of the length of the train.
- Each train has a fix speed of 100 m/sec while passing the tunnel and length of tunnel is 100 meters.
- A train arrives to point A, E, or F from other parts of the city with probability p at every second. However, a train arrives to point B with probability $1 - p$. Make p a command-line argument.
- A train arriving to any point from other parts of the city has an equal probability of running for any of two different lines after passing the tunnel. For example, a train

arriving to point A can be running for Line-1 or Line-2 with equal probability, or a train arriving to F can be running for Line-2 or Line-4 with equal probability.

- A train can have the length of 100 meters with probability 0.7 and 200 meters with 0.3.
- For efficiency, the Metro control center gives permission to pass the tunnel to a train from the section that has the largest number of trains.
- If there is a tie, the Metro control center gives the following priority to trains waiting at: A > B > E > F.
- At time zero, there is no train in the system.
- Add a command line argument $-s$ to indicate the total simulation time (e.g. -s 200) in secs.

Part II

(20 points) Part I may cause starvation to the trains that have a lower priority because of their departure point. We will have the following solution to prevent starvation.

- System overloading happens once there are more than 10 trains in the whole system except for the section containing tunnel. In this case, Metro control center notifies the trains from other parts of the city to slow down. As a consequence, no new train arrives to A, B, E and F until all the trains clear the tunnel.

Part III

(10 points) Once in awhile, a train may breakdown while it is in the tunnel with 0.1 probability, which will result in 4 additional secs for the train to pass the tunnel.

Keeping Logs

(20 points) You are required to keep a log for the trains and for the Metro control center, which will help you debug and test your code. The train.log should keep the train no (ID), its starting point, its destination, length, arrival time, departure time of the system. The control-center.log should keep the events; which train no (ID) passes the tunnel at what time, whether it broke down or not, whether system overloading happened, whether train clearing event happened and how long it took to clear up the tunnel section.

train.log:
Simulation Arguments: p=0.5, ...=..., (other optional arguments)

Train ID	Starting Point	Destination Point	Length(m)	Arrival Time	Departure Time
0	B	E	100	18:21:23	18:21:27
1	F	A	200	18:21:37	18:21:46
2	A	E	100	18:21:38	18:21:48

control-center.log:

Event	Event Time	Train ID	Trains Waiting Passage
Tunnel Passing	18:21:28	0	
Tunnel Passing	18:21:38	1	
Breakdown	18:21:40	1	2
Tunnel Passing	18:21:45	2	3, 4, 5
Tunnel Passing	18:21:47	4	3, 5, 6, 7, 8, 9
Breakdown	18:21:48	4	3, 5, 6, 7, 8, 9, 10, 11
System Overload	18:21:49	#	3, 5, 6, 7, 8, 9, 10, 11
Tunnel Passing	18:21:53	6	3, 5, 7, 8, 9, 10, 11, 12, 13, 14
Tunnel Passing	18:21:57	3	5, 7, 8, 9, 10, 11, 12, 13, 14
Tunnel Passing	18:21:59	5	7, 8, 9, 10, 11, 12, 13, 14
Tunnel Passing	18:22:01	8	7, 9, 10, 11, 12, 13, 14
Tunnel Passing	18:22:04	7	9, 10, 11, 12, 13, 14
Tunnel Passing	18:22:03	10	9, 11, 12, 13, 14
Tunnel Passing	18:22:05	12	9, 11, 13, 14
Tunnel Passing	18:22:07	9	11, 13, 14
Tunnel Passing	18:22:09	11	13, 14
Tunnel Passing	18:22:11	14	13
Tunnel Passing	18:22:14	13	
Tunnel Cleared	18:22:16	#	# Time to Clear: 27 sec

Suggestions

- You may want to keep a queue for sections A-C, B-C, D-E and D-F, and add trains 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 trains, specify lines and lengths at the specified probabilities. For easy debugging, add a command line argument for a seed and feed the seed to the random number generator.
- One way to implement the simulation to represent each section as a thread. The control center should have its own separate thread.
- Start simple. For example simulate two sections first (A-C and D-E). Add other sections and complexity as you are sure the current implementation works (no deadlock, no accident).
- If you need 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=0.5.
- any supplementary files for your implementation (e.g. Makefile)
- a README file briefly describing your implementation, particularly which parts of your code work.
- Finally your team will perform a demo to TAs after the project submission deadline.

GOOD LUCK.