Homework #8

Due 03:00pm on Tuesday 04/05/2016 through gsubmit.

Feel free to make assumptions, if you feel that such assumptions are justified or necessary. Please state your assumptions clearly. Unreasonable assumptions lead to unreasonable grades!

---

1. *Tunnel Synchronization.* There is a long windy tunnel through a remote mountain that needs to be repaired, necessitating the closure of all but a single lane, which must now be shared by vehicles traveling in both directions. If two vehicles going in opposite directions meet inside the tunnel, they will crash causing the whole tunnel to collapse and the end of the world as we know it :) Furthermore, given the lack of good ventilation, the tunnel is safe to use as long as there are no more than N vehicles traveling inside the tunnel. Design a synchronization protocol with the following properties:

    - Once a vehicle enters the tunnel, it is guaranteed to get to the other side without crashing into a vehicle going the other way.
    - There are never more than N=4 vehicles in the tunnel.
    - A continuing stream of vehicles traveling in one direction should not starve vehicles going in the other direction.

    Implement your protocol using Java two types of threads representing vehicles going in the two directions, respectively. Explain the purpose from any semaphore/variables in your solution and its initialization. Sow that your code works by having a set of threads (say 10 or 20) repeat the following forever: (1) pick a direction for travel, (2) wait until it is safe to enter tunnel, (3) travel inside tunnel for some amount of time, (4) get out of the tunnel on the other end. To test your protocol, you may want to experiment with various parameters (e.g., make one direction much more popular than the other to make sure that starvation does not happen, change N, etc.)

2. *Shuttle Synchronization.* Logan Airport has upgraded its fleet of shuttles serving its terminals to driverless electric shuttles.

    The shuttles simply go around the airport in a circle from terminal i to terminal (i+1) mod k, where k is the number of terminals to be served (K= 6 for Logan, given the stops at terminals A, B, C, D, and E, and at T station). Each shuttle has a fixed capacity N, i.e., no more than N travelers can ride on the bus. When a shuttle arrives to a terminal stop, some of its riders may leave while others may board up to the capacity limit. To discourage travelers from rushing to catch a shuttle that is already loading its passengers, anyone arriving to the shuttle terminal stop while the shuttle is boarding must wait for the next shuttle. In order for the shuttle to close its doors and leave the terminal, some passenger must tell it to do so. To save energy and avoid unnecessary stopping, the shuttle should not stop at a terminal stop unless at least one passenger needs to get on the shuttle or off the shuttle at that stop.

    The passengers in this setting repeat the following steps forever: (1) pick a starting terminal, (2)

wait for shuttle, (3) board shuttle, (4) pick a destination terminal, (5) wait for shuttle to arrive at destination, and (6) get off the shuttle.

Design a protocol that works for a single shuttle and for any number of passengers. Implement your protocol using Java threads, allowing the shuttle thread to interact with passenger threads. You should assume N=10 and K=6, and you should assume that there are 50 passengers in the airport. You should also assume that the shuttle thread spends some amount of time to go from one station to the next (by having the thread sleep for some random amount of time). Show that your code works as specified above and that it is free of deadlocks.

3. *Shuttle Fleet Synchronization.* Adjust the protocol you developed for Shuttle Synchronization to allow the airport to operate M > 1 shuttles. Clearly, a safety requirement is that there would never be more than a single shuttle loading or unloading passengers at any one terminal. Explain any new synchronization you introduce to your solution and implement your changes in Java and show that it all works as expected.

4. *Self-Charging Shuttle Fleet Synchronization.* Shuttles need to recharge their batteries periodically. The shuttle keeps track of the amount of time spent traveling between terminals (the sum of all the random sleep times). Once that time reaches some threshold, it is time for the shuttle to be recharged. For example, if the average travel (sleep) time between terminals is 5, then the threshold could be set to 100. When the threshold is reached, no new passengers may get on the shuttle, and once the last rider gets off, the shuttle would get to a charging station, spend some time being charged, and resumes its normal operation after that. Explain any new synchronization you introduce to your solution and implement your changes in Java and show that it all works as expected.
