EE 360P, Sample Test Problems, Vijay K. Garg,

Q.1 **(20 points)** Write a Java class `PLock` that provides priority based locks. The class provides the following methods `requestCS(int priority)` and `releaseCS()`. The priority is either 0 denoting low priority or 1 denoting high priority. Whenever the critical section becomes available, a thread that is waiting with the highest priority is given the access. If there are multiple threads with the same priority level waiting for the critical section, then any one of them may enter the critical section. You should guarantee that a low priority thread does not enter the critical section if a thread with a higher priority is waiting.

Q.2 **(15 points)** Give Java class for `Latch`. A Latch is initialized with a given count. It supports two methods, `await()` and `countDown()`. The `await` methods block until the current count reaches zero due to invocations of the `countDown()` method, after which all waiting threads are released and any subsequent invocations of await return immediately. This is a one-shot phenomenon – the count cannot be reset.

Q.3 **(15 points)** Implement the class `myLock` to coordinate entry into a room by various threads. There are two types of threads in the system - type1 and type2. The only requirement is that there should not be two threads of different types in the room at any time. Implement the class `myLock` with three methods `void request1()`, `void request2()` and `release()`. Threads of type 1 use `request1()` and threads of type 2 use `request2()` to enter the room. All threads use `release()` to exit the room. You do not have to worry about starvation.

Q.4 **(20 points)** An old bridge has only one lane and can only hold at most 4 cars at a time without risking collapse. Create a monitor with methods `arriveBridge(int direction)` and `exitBridge()` that controls traffic so that at any given time, there are at most 4 cars on the bridge, and all of them are going the same direction. A car calls `arriveBridge` when it arrives at the bridge and wants to go in the specified direction (0 or 1); `arriveBridge` should not return until the car is allowed to get on the bridge. A car calls `exitBridge` when it gets off the bridge, potentially allowing other cars to get on. Don't worry about starving cars trying to go in one direction; just make sure cars are always on the bridge when they can be. You must use Java ReentrantLock and Condition variables. *Ensure that you signal only the appropriate cars when you exit the bridge.*

Q.5 **(20 points)** Write a Java class `FifoLock` that provides the following methods `getTicket(), requestCS(int ticketnumber)` and `releaseCS()`. When any thread invokes `getTicket()`, it gets a ticket number. You should ensure that no two ticket numbers are identical. After getting a ticket number, the thread can invoke `requestCS` anytime. It must provide the ticket number that it received using `getTicket`. Assume that threads do not cheat and provide the genuine ticket number. You should guarantee the following property. *Any process that invokes* `requestCS` *enters the critical section if the critical section is empty and there is no process with lower ticket number waiting in the queue.*