Homework 7

Erik Brakke

March 29, 2016

Answer 1

This code will work
The scenarios here are the either a process gets a ticket before another, or one gets interrupted and they both happen to get the same ticket number
In the case where one process gets a ticket before the other, then all other processes that came after will have to wait on that process to finish (i.e. the ticket = 0)
If they both have the same ticket number, then there is a tie breaking scenario and the process with the lesser ID will go. This ensures that both will not go at the same time, and the other process will go once the first has set its ticket to 0

uncomment the code for part 1 in Simulation.java to see the results

Answer 2

1. This code works because ever processes is effectively waiting on every other process to arrive. When one arrives, it signals, which means that anyone else waiting can then signal that process i has arrived. No process will be able to go until all proceeses are there, because the initial signal comes from process i. Each semaphore should be initialized to 0

2. No, because after the rendezvous, semaphore[i] will always be 1, therefore the code will not block. The extra wait(semaphore[i]) is called to ensure that the semaphore goes back to 0

Answer 3

1. See code. R is an array the keeps track of priority for processes that are requesting the CS
   N is the number of processes currently requesting the CS
   B is an array of binary semaphores

2. Uncomment 'Question3b' in Simulation.java to see output

3. Uncomment 'Question3c' in Simulation.java

4. Uncomment 'Question3d' in Simulation.java

5. Starvation is possible in part A. This is beause high priority requests will always beat low ones, so if there are a lot, low priority requests will not go. In B and C, there is no possibility of starvation. In part C, there is a bound of 0 out of order (because it is FIFO). On part A there is no bound (because starvation is possible).