Summary - Project Two

Kaushik Nadimpalli
CS4348.001

Purpose and Project Overview:
The purpose of this project is to study the coordination of multiple threads. We had to simulate a hotel by using threads to model customer and employee behavior. The simulation has two employees at the front desk to register guests and two bellhops to handle guests' bags. A guest will first visit the front desk to get a room number. The front desk employee will find an available room and assign it to the guest. If the guest has less than 3 bags, the guest proceeds directly to the room. Otherwise, the guest visits the bellhop to drop off the bags. The guest will later meet the bellhop in the room to get the bags, at which time a tip is given. There are a 25 threads for guests, 1 for each. There are two threads for front desk employee(2 of them) and 2 threads for bell hops(since there are two bell hops). Semaphores were used in this project to achieve mutual exclusion and coordination. Specifically, I utilized 12 semaphores, three of which are mutex(s), and 2 are semaphore arrays. For mutex(s), the same thread that acquires it must release so I programmed the wait and signal calls in the same function. Most of the semaphores I used are binary semaphores, but I did have two counting semaphores as well. This is for the front desk employee and the bell hop since there are two of them each, and each should be able to hold those semaphores simultaneously since they will be helping different guests. With this project, I did face a few difficulties with the coding structure and finding a way to implement the simulation.

Learning curve and difficulties faced:
Some of the difficulties I faced while I was going through this project was figuring out how many semaphores were exactly needed and the exact positioning of the wait and signal calls. At first, I had trouble coming up with the pseudocode and I had to closely follow the example on the slides about the Barbershop to get an idea on how to structure the hotel simulation. I decided to encapsulate all the threads and their activities in their own methods. The main will simulate and print only when guests are joined. This worked well since it printed each threads activity very clearly on the command line as opposed to doing the whole project in one single class. Another struggle I ran into was finding out more about the java.lang.Thread package and how semaphores worked in Java. I also found out more about the Runnable interface, whose instances are intended to be executed by a thread. The class must define a method of no arguments called run. Most of the code related to the semaphore calls - acquiring and releasing them in Java - was done in the run method for my program. Lastly, I also had trouble printing out when a guest has joined the hotel after they retired for the evening. I did this in the guest class in a finally block. I had a lot of try catch blocks in the program, so I wanted to ensure that this block is executed even if an unexpected exception occurs. I personally believe that working on the pseudocode, especially the functions with the wait and signal calls, really helped my understanding

of how the entire simulation works. It was a good learning experience since it expanded my knowledge on how multiple threads coordinate to simulate a producer consumer scenario.

Summary - Overall experience
Overall, I had a great experience working through this project. There is a huge difference in conceptually understanding how semaphores work and actually implementing them in a programming language. In this project, firstly, I divided the program into different classes. The instructions were clear that each thread(represented by the class) must print its own activities. The semaphores were written outside any of the subclasses, since they will be accessed - waited on or signaled to - by more than one of these classes. I have three classes and multiple thread instances are built in each. I used try and catch blocks, and most of my functionality is in the try block since the simulation might throw an exception and that exception should be caught in that case. One thing I wish I could've done better was consider security when designing a simulation such as this. Almost all of my variables are static member variables, but I should've instead used instance member variables. Another logical flaw that might become a security issue is that I created and initialized the semaphores in the main superclass. I should've don't this within the methods themselves since there might be encapsulation issues.

Results - output(entire simulation)

The results for my program were similar to the output. The simulation first creates the two front desk employees and then the two bellhops. It then creates the guests and the process runs. Below is a sample output until Guest 5 is created for my program. The program runs till all guests 0-24 are joined, this is just a small part of the actual output.

```
Simulation starts
Front desk employee 0 created
Front desk employee 1 created
Bellhop 0 created
Bellhop 1 created
Guest 0 created
Guest 1 created
Guest 0 enters the hotel with 2 bags
Guest 2 created
Guest 1 enters the hotel with 0 bags
Front desk employee 0 registers guest 0 and assigns room 1
Front desk employee 1 registers guest 1 and assigns room 2
Guest 0 receives room key for room 1 from employee 0
Guest 0 enters room 1
Guest 1 receives room key for room 2 from employee 1
Guest 0 retires for the evening
Guest 0 joined
Guest 1 enters room 2
Guest 1 retires for the evening
Guest 2 enters the hotel with 1 bags
Guest 1 joined
Guest 3 created
Front desk employee 0 registers guest 2 and assigns room 3
Guest 2 receives room key for room 3 from employee 0
Guest 2 enters room 3
Guest 3 enters the hotel with 2 bags
Guest 2 retires for the evening
Guest 2 joined
Guest 4 created
Front desk employee 1 registers guest 3 and assigns room 4
Guest 3 receives room key for room 4 from employee 1
Guest 3 enters room 4
Guest 3 retires for the evening
Guest 3 joined
Guest 5 created
```