Ajk190003
Adam Kosicki

Project 2 Summary

Design Process

For my DMV simulation, I started by coding rather than doing the design. I decided to do the coding first because I like to experiment with new methods and overall I learn a lot better and faster with hands-on experience. After I felt comfortable with semaphores and how they work in Java, I began by initializing and starting the threads in the main method. This is the first step to making use of the semaphores. Next, I made the necessary classes, which are Customer, Agent, InfoDesk, and Announcer. Knowing that there are going to be multiple Customers and Agents, I made sure to add an ID within the classes so I could identify the different threads while the code is running. The information desk worker and the announcer do not need a way to identify themselves because there is only one of each. The customer class is the basis of the project, they start the DMV process so I began with them.

Simulation

The information desk releases a signal that the customer class receives to know that the information desk is available. Once that happens, the information desk sets a global variable to a number, then increments it. The customer class grabs the number and stores it, then sends a signal representing them receiving their number and entering the waitroom. Once they are in the wait room, they enter the “waitRoom” queue. Then, the customer sends a signal to the announcer with the “customerEntersWaitroom” semaphore. The announcer class then makes sure that the other queue, which is named “agentLine”, has less than 4 individuals in it. If there are less than 4 people in the queue, then the announcer calls the value “currentNum” from an array of semaphores called “numberCalled”. This notifies the customer with the number that was called, that their number was in fact called with that list of semaphores. The “currentNum” value increments, and the customer is added into the agentLine queue, and then they walk to the agent line via the “walkingToAgentLine” semaphore. The agent then waits for a customer by waiting for the “agentAvailable” signal. Once the signal is received by one of the agents, the customer removes himself from the agentLine queue, and the agent prints out that they’re serving that specific customer. Then with the “agentAsks” semaphore, the customer is notified to get their photo taken and take their eye exam. Lastly, the customer signals the agent that they are done, so the agent signals the customer back with the “giveCustomerLicense” semaphore so the customer gets their license and leaves with the “customerHasLeft” semaphore. The customer is then joined at the end with a for loop at the end of the main method that waits for all the customers to be joined, prints “Done” and exits the code.

Difficulties Encountered

I had the most issues with global variables within the project. There has to be a way to communicate to the agent that they are serving a specific customer so they can print that out. The issue with that is the fact that both agents are using the same class, but could be using the same global variables at the same time. I ended up using semaphores to make sure that a specific global variable is only being used by a specific class by one thread at a time. It does not require the use of anything like sleep or busy waiting.

Learned

I learned a whole lot with semaphores and how useful they are. I am planning on using them to make sure that an action can occur once something else is done. I also connected them to a hackathon that I worked on the previous weekend, where we were looking at data within a car. I noticed that a lot of values were changing from 1 to 0 repeatedly, and I now understand that it is not all random. There are bits of code that coordinate this, for an example, to make sure a car’s window is currently opening.

Results

My results were successful, since my code works as intended. All the threads communicate to each other correctly no matter the amount of customers that I add. This means that there is no way for my code to be deadlocked.