Summary

Simulation Architecture

The program is started from the Main thread, named Theater. This Theater holds the Semaphores, as well as queues which are used for inter-thread communication and coordination. To make it easier to program, the individual threads extend this Theater class. In this way, they are able to directly access the Semaphores and other coordination variables without having to call functions. Each individual thread also implements Runnable, so that they can be started from the main thread.

The Theater thread simply initializes the objects needed for the simulation, including Semaphores and queues and starts the individual threads needed for the simulation. Then the program waits on a Semaphore for each customer (found in an array of Semaphores indexed by customer ID) to be signaled. When this is signaled, it indicates that the customer has joined the main thread. When all of the customers have signaled their semaphores, the main thread ends the simulation, bringing down any remaining threads (such as the box office agent).

The work-flow for a customer seems to be fairly consistent, so it is not surprising to see many repeated protocols in the communication system. When interacting with a worker, a customer needs to tell a worker some information, including their ID and some item of interest (movie title, snack type, etc.). To facilitate this transfer of information, it is packaged up into a container class. This class is then enqueued by the customer and dequeued by a worker. There is a control semaphore to ensure that only 1 entity can access this queue at a time. After the customer has enqueued their transaction, they signal another semaphore to indicate that they are ready. Then they wait on a Semaphore signaling that the worker has processed their transaction. This semaphore is located in an array indexed by customer number. This pattern is present in the interactions with the BoxOfficeAgents, TicketTaker, and ConcessionStandWorker.

Programming Difficulties

This program was pretty smooth since I planned out the pseudocode and all semaphores before I implemented. I was able to basically just copy and paste my pseudocode and almost everything went as planned. However, in my initial design, I forgot to release the readMovie semaphore if a customer got to the box office agent but the movie was sold out. This lead to a race condition, as the customer thread who left is still holding the readMovie semaphore, so no other threads can access the Movie file or list in any way. After realizing my mistake, I was able to fix it in a trivial amount of time.

What was learned

Before this project, I had used semaphores in C and C++, but this was the first project where I had used them in Java. So one thing I got out of this project was experience using Java semaphores.

Results

Before I started this project I did not have a very firm grasp on semaphores, even though I had used them in previous projects. After sitting down and designing the system, I

was able to visualize the communication process, and this helped me get a better understanding of semaphores. Furthermore, my experience with this project, especially the general ease of the experience, has given me the confidence I need to successfully use semaphores in future Java applications.