Naman Gangwani
nkg160030@utdallas.edu

CS 4348.002
March 31, 2018

Project 2 Summary

This project focuses on simulating a simple clinic with a single receptionist and an arbitrary number of patients, nurses, and doctors, where there may be up to 30 patients and three nurses and doctors (with each doctor having his/her own corresponding nurse). The simulation is based solely on threads and semaphores with no other methods of thread manipulation and blocking being employed. It relies on mutual exclusion and coordination amongst the multiple thread representations of the patients, receptionist, nurses, and doctors through semaphores. Each person entity must be its own thread, and there must be minimal mutual exclusion (to allow for the most concurrency) and no thread sleeping as a mode of coordination. Inside the simulation itself, an individual patient must first talk to the receptionist, where he/she will register the patient. Then, that patient should head over to the waiting room where a randomly assigned nurse/doctor pair will coordinate with him/her afterwards. The randomly assigned nurse will take the patient in the waiting room to his/her corresponding doctor's office when he/she is available, and the patient will interact with the doctor (such as explaining symptoms, listening to advice) until he/she is free to leave. Throughout this entire patient/receptionist/nurse/doctor process, there may be multiple patients who need to be served by the clinic, so everything must work concurrently to get all the patients attended to by the doctors in the most efficient manner possible.

Throughout the process of implementing this project in the Java programming language, I didn't have too many difficulties when actually writing the program itself, but I did have to spend some time rethinking and redesigning my semaphore design and pseudocode when writing out the design for this project. Initially, I was trying to handle the wait and releases oppositely (such as the patient being blocked until the receptionist unblocks it) until I found that it is more efficient to handle waiting and signaling in a different way. Additionally, I was having a hard time wrapping my head around how I would specifically represent which patient is assigned to which nurse and doctor. That is why, when writing out my program, I decided to utilize an array called *currentPatientOfDoctor* that kept track of the ID of the patient that it was assigned to, with a value of -1 if the doctor did not currently have any patient. Similarly, an array of semaphores for nurses, doctors, and the patient-doctor interactions allowed me to create an efficient "mailbox" type communication between the patient and the nurses/doctors, which was something that I was having trouble on deciding how to do when I first started out.

Knowing nothing about semaphores prior to this class, I felt like I learned a lot about them with this project. I understood the concepts required in creating coordination amongst multiple threads, such as blocking and signaling with as minimal mutual exclusion as possible. There were many instances where I subconsciously wanted to utilize busy polling, but then I realized that there are better methods and found workarounds through semaphore coordination, and I ultimately didn't have to utilize it at the end. Overall, it gave me a better understanding of the importance of CPU time and resources.

The result for this project's program seemed to be working exactly how I intended. The design looks proper, the program looks well-written and organized with concepts of Java abstraction in it, and the output performs the simulation with the perfect amount of concurrency for any arbitrary number of patients, nurses, and doctors.