Sammy Pandey
SXP210252
CS 4348.501

Project 2 Summary

Threads and Semaphores

This second project in Operating Systems explored the concepts of threads and semaphores, and the coordination of multiple threads using semaphores. It simulates the general events that happen in a health clinic, with the main participants being Patient, Receptionist, Doctor, and Nurse. Each of these characters have their own thread. The number of doctors and nurses are the same, and it is specified by the first command line argument. The number of patients to be run is specified by the second command line argument. Each patient thread will enter the clinic knowing the doctor they would like to see, which is a random integer from 0 (inclusive) to the total number of doctors (exclusive). They will then wait to register with the receptionist, and then go and sit in the waiting room. The receptionist alerts the nurse working for the requested doctor that a patient is waiting. The nurse takes the patient to the doctor's office if it is free, and tells the doctor that the patient is waiting. The doctor then visits the patient, listens to their symptoms, and advises them. The patient leaves, and it continues concurrently until all patients have been seen.

My language of choice was Java, since I wanted to learn how to implement threads and semaphores in a language other than C. I first had to reteach myself basic java code, and also learn how threads and semaphores work in Java, since my Unix class did not cover Java threads. The hardest part was knowing where to start, as each thread running concurrently means there is no true starting and ending point. I started with the design portion, first by writing out the steps and the data structures I would need for each thread to communicate with each other. I decided that my best option would be to use BlockingQueues, as well as Arrays of Semaphores and Arrays of Blocking Queues.

Some issues I ran into was figuring out the exact job of the Nurse thread, and who to give the task of placing the patient on a queue. I figured out that the Nurse should have a queue of patients, but the doctor does not need a separate one, as they will only see one patient at a time. So, I tasked the receptionist with passing the Patient ID to the corresponding Nurse Queue, and then made the Nurse check whether their doctor is free before removing them from their personal queue. The Nurse would also have to have access to the corresponding doctor object to pass it the Patient's ID. Another major issue I ran into was that the Nurse would take the patient into the office before the patient had left reception. This, along with most of the issues I encountered, was resolved by adding another semaphore that alerts the Nurse that the Patient they removed from their queue is in the waiting room. The final issue I ran into was figuring out how to end the program, which I did by keeping a count for every time a patient exits the office. This was compared with the total number of patients specified to know when to exit the entire system.

In conclusion, I learned a lot through the implementation of semaphores to simulate a visit to the doctor's office. It required many semaphores and arrays of semaphores to control thread concurrency, as well as the practice of locating error with multithreading.