Project 2 Summary

The purpose of Operating Systems Project 2 was to create a simulation of a doctor’s visit by coordinating various threads with only semaphores. The threads involved where a receptionist thread, up to thirty patient threads, up to three nurse threads, and up to three doctor threads where the number of patient and nurse/doctor threads were command line parameters. The simulation begins with a patient entering the waiting room, waiting to speak with the receptionist and being randomly assigned a specific nurse/doctor pair. Then the receptionist acknowledges the patient and registers the patient then the patient sits in a waiting room and the receptionist alerts the assigned nurse to get the patient from waiting room. The nurse then retrieves the patient and alerts the doctor that the patient is there. The patient then enters the office and tells the doctor their symptoms and the doctor listens to the patient and then proceeds to give advice. The patient listens to the advice, leaves the office, and alerts the nurse/doctor pair that the patient has left so the pair may tend to a different patient.

In this project, I would say the use of semaphores as the only means of thread coordination was a challenge at first, but I quickly realized using semaphore arrays with unique IDs were essential as they allowed to wait and signal to specific patient/nurse/doctor threads. Before this, I was trying to do the project with solely regular semaphores which deadlocked after using multiple patient threads at the same time. Another challenge I encountered was a C++ specific issue where my program had threads that would overwrite each other on the same line so if a patient entered at the same time that a nurse calls a patient to a doctor’s office, the two lines would essentially merge and write on top of each other. This issue was my main challenge of the project as I spent many hours foolishly using “cout” print method instead of “printf” print method. After many google searches, I found printf was thread safe and my issues were immediately solved upon changing all cout statements to printf statements.

In this project, I learned a great deal about how to create threads, manage them using semaphores, mutual exclusion, and how to ensure that specific threads are signaled using semaphore arrays with their unique IDs. Prior to this project, I barely knew anything about thread programming but after this project, I feel more comfortable about the subject after this project. I would say the use of only using semaphores for coordination and mutual exclusion was also very beneficial as it required a great deal of thinking about the barbershop design instead of cheating the design by using busy/sleeps calls. As far as mutual exclusion in the program, I kept it minimal where the areas of mutual exclusion included registering one patient at a time and only allowing a nurse/doctor pair with a specific ID to tend to one patient at a time which means nurse/doctor 0 can not tend to 2 patients at the same time.

The results of my project 2 ended in a successful running program that can handle up to 30 patient threads, 1 receptionist thread and 3 nurse/doctor threads that only use semaphores for thread coordination. In addition to semaphores and threads, a popular data structure in my program were queues and hash maps. The first queue I used was to keep track of which patient thread entered and which patient threads were added to their respective doctor queue. Another queue that I used was called a finishedQ that would essentially be used to keep track of when patients threads exited. So when a thread exits, it would push its unique patientID to finishedQ and when finishedQ had the same number of elements as the inputted patient parameter which means all patient threads have finished execution then the main method will exit and end the remaining receptionist, nurse, and doctor threads. And the last data structure that I utilized in the project was a HashMap in order to keep track of which patient is assigned to which doctor so the HashMap would keep track of the patient to doctor relationship.