CS4348.006 Zach Tang Project 2

Summary

In this project, we simulate a hotel with three actors: one to twenty-five guests, two desk employees, and two bellhop employees. I created a function for a guest, a front desk employee, and a bellhop employee. These functions are run on the threads that correspond to the actors. I create a thread for each actor in the hotel simulation. For example, if we have five guests, then I create five threads with an independent guest function running on each. In the functions of each actor, there is code that interacts with other actors of the hotel. I use semaphores to ensure that no deadlock occurs where actor A is waiting on actor B for an action while actor B is waiting on actor A for an action. For example, I use guestDeskQueueMutex to ensure that only one thread can access the queue at a time. Otherwise, both of my front desk employee threads could access the queue's front at the same time. Then they would both service the guest at the front. This would definitely lead to deadlock as both employees would wait on the guest's semaphore signal that the guest left the front desk; however, this signal would only free up one employee thread, leaving the other blocked forever. With the use of semaphores, I successfully simulate a hotel's operations.

There were some difficulties encountered in this project. My program would run into deadlock sometimes even though I had thought I designed the semaphores in such a way to avoid that. In these cases, I used the debugger gdb to inspect the state of my program. gdb has a great feature "info threads" that let me inspect the state of my threads when my program was in deadlock. This feature displayed all the blocked threads and the semaphore each was waiting on. This allowed me to see the exact resources deadlocked. I then referenced where those resources were called in my code to develop specific execution cases where deadlock would occur. After coming up with exact scenarios where deadlock would occur in my code with the help of gdb, I was able to fix my code to address those scenarios. For example, my guest threads and front desk employee threads were deadlocked. I used gdb to visualize the specific deadlock occurring. Two guest threads were blocked on the deskEmployees semaphore, while the two front desk employee threads were blocked on the desk[] semaphores. I referenced this specific area in my code where the guests waited on the deskEmployees and the front desk employees waited on the desk[] semaphores. I realized that because I had the guests wait on the deskEmployees semaphore *after* queueing up, a guest could be served by the front desk employee while being blocked by the deskEmployees semaphore. Therefore, the front desk employee would never get a signal from the guest that they left the desk because the guest was blocked. Moving the guests to wait on the deskEmployees semaphore *before* queuing up fixed this deadlock scenario.

Another difficulty occurred with output. I originally didn't have a printMutex semaphore, so the output of my program became interleaved and jumbled. To solve this, I created a printMutex semaphore with an initial value of one. Then, threads that wanted to print would have to wait on the printMutex to become available. Because the printMutex is initialized to one, only one thread would be able to print at a time. After that thread is finished printing, it signals the printMutex semaphore to unblock the next thread waiting.

I learned a lot from the difficulties faced in this project. I became more familiar with the gdb bugger and learned the information provided in "info threads" and how to use it to debug deadlock scenarios in multithreading environments. I also learned how to lock a specific resource with semaphores so that only one thread can access it at a time. An example of this is the printMutex or the guestDeskQueueMutex semaphore. Overall, this project was very enjoyable for me and the results were great. I created a python script to run 1250 simulations of the program and to verify each output of those 1250 simulations. This script helped me catch faults in my program (e.g. the guest entering their room before making sure the bellhop employee took their bags). After fixing those faults, all 1250 simulations passed the verification.