CS 4348 – Operating Systems Project 2 Summary

This projects purpose was to study possible concurrency issues and how semaphores interact with each other. We simulated a hotel, with *25* guests, *2* clerks, and *2* bellhops, each using their own thread. In my approach I went with a more object-oriented workflow and created classes for the guests, clerks and bellhops. For my semaphores, I created a global hashmap (for fast lookup and readability when accessing them) to handle global interactions between the threads, and I created separate relative hashmaps for each clerk-guest and bellhop-guest interaction. Per the instruction, each guest interacts with the clerks based on their talking criteria (handled by the relative hashmaps), as well as the bellhops. I encountered some very unpleasing issues in this project. For one I had to rewrite my code base about four times to get a solid less-finicky framework to work with. I also spent about four days trying to solve an issue with my code involving race conditions only to find out that my issue was in fact NOT a race condition, but an issue within the design of how my classes interacted (I’m not frustrated at this point I swear). I failed to realize how important mutexes (binary semaphores) were and how they helped enforce mutual exclusion among critical sections until about 3 days into the project, unto which after I added them I began to see little to no problems within my output. My last problem was trying to get my guests to enter and interact in a predictable order (i.e. if guest one enters first he is first in line compared to guest 2, etc…), so I added a queue to handle those interactions. Overall, I learned that concurrency was very annoying to deal with and is something that should be of large concern when building applications that have concurrent operations. Testing is a must and solidifying the logic before even starting to code is extremely important. Something strange I noticed (not that strange if you think about it) was how increasing the number of threads created different output (during testing). For example, just because something ran correctly with seven threads doesn’t mean that it will run correctly with twenty-five threads. This can be attributed to processor speeds I’d assume, since faster processors wouldn’t have as many race condition problems to get hung up on. Overall the project was a real pain, but it was interesting to see the interactions between semaphores and how they all interacted together.