Cameron Meyer

CS 4348.001

Project #3 Summary

In Project #3, I was tasked with implementing several scheduling algorithms to simulate running jobs on a system. These algorithms included First Come First Served (FCFS), Shortest Process Next (SPN), and Highest Response Ratio Next (HRRN). Each of these scheduling methodologies were non-preemptive, so I didn’t have to worry about one job being able to interrupt another. Instead, I could focus simply on scheduling an entire job at once, and then output that in a graphical, but text-based format.

Project 3 felt fairly straightforward, and easily achievable within the allotted time. I felt confident about this project throughout the whole process, and didn’t hesitate to dive into some pseudocode right after reading through the project description. Referencing the lecture slides also helped immensely to ensure I understood the concepts behind each scheduling algorithm. At this point, I’ve also become more used to the necessary development environment for the course. Again, I completed development for this project in Visual Studio code. From this IDE, I was able to SSH directly into UTD’s cs1 Linux server, where I could run and test my code directly from the proper environment. This helped me to ensure I would have no strange or unexpected bugs occurring between a local environment for myself and the cs1 environment after submitting. Throughout development, I would periodically save my work after completing a new feature, and commit and push those changes to a public git repository. The VS Code terminal allowed me to commit my changes from the command line.

I decided to implement project 3 in C++, to further refine my skills with the language and challenge myself. My program expected command line input from the user so it could determine which input file to read job information from. This user input is handled by the program to ensure the file exists before accessing data from it, and it will error out if given an invalid file. I also spent time developing helper functions for this project, like sort functionality so I could organize my vector of jobs based on run duration, and a custom print function to output my schedules properly. The schedules were assembled within each scheduling algorithm, and they output an ‘X’ to represent a job that is currently running, or a space if a job is not running. These scheduling lines are added to a vector, and passed to the print function to output the correct schedule for the associated job.

In order of difficulty, I found implementing FCFS to be the most straightforward, followed by HRRN and then SPN. Since jobs in the input file were already organized by arrival time, it made implementing the first algorithms much simpler to understand. When implementing these, I really didn’t encounter many bugs either since I’ve become more used to handling file I/O over time, and the processing and output was pretty manageable. The bugs I did encounter were all easy to fix, and usually associated with improper file parsing, or using the wrong variable scope within a loop of my scheduling algorithm. Pseudocoding out my algorithms in advance also worked wonders to help me prevent bugs as much as possible.

While this project doesn’t perfectly simulate the arrival of jobs in real-time, and its small scope doesn’t allow for practicing preemptive scheduling algorithms, I still enjoyed my time completing the assignment. Overall, it was a neat exercise in practicing logic and processing for jobs in an organized way. There’s more that could be done to expand the scope of the project for future semesters, though given how late we are into the semester and how many other tasks and exams students are juggling, this felt like a much more easily digestible project than the earlier projects in the course.