Naman Gangwani
nkg160030@utdallas.edu

CS 4348.002
April 23, 2018

Project 3 Summary

This project focuses on simulating a scheduler that schedules a given set of jobs based on their arrival times and durations with six different algorithms. I approached this project first by laying out all the basics such as handling the file reading and creating all the necessary functions that will be of use to make this project be as close to as functionally based as possible. I wanted a neat and organized way of handling all the data given to the program in the file, so I decided to utilize a 2D array containing the jobs' arrival times and durations, with the indexes representing the jobs' alphabetized character values. Once this array was populated with the file's data, I could then pass it over to the appropriate algorithm's function based on what the user specified in the command line. From there, the function could take over and perform its respective algorithm using the data in the 2D array passed to it. In total, there would be six algorithm functions. In each function, there would be a similar structure of a do-while loop that executed until all of the jobs ran to completion. Additionally, there would be queues, arrays keeping track of the number of seconds left for each job, variables indicating what the current job running is, and idle time handling methods within each function. What would differ in each algorithm-based function is how exactly the algorithm specific to each function is handled and executed, which would involve picking the next valid job for execution and under what circumstances to do so. Finally, certain helper functions such as printing the alphabetic characters representing the jobs and printing the 'X' that specifies which job is running were created for organizational purposes to reduce the program to be more functionally based.

I did not find anything in this project particularly difficult, but I did encounter some hiccups in the way when trying to complete it. When working with queues in C++, popping from a queue would not return the value that was popped, so workarounds had to be made by saving the front of the queue to a variable before popping from it. Additionally, working with numerical limits such as INT_MIN and INT_MAX would work fine in an IDE environment, but they required the *#include <climits>* statement when working in a UNIX environment. Certain precautions had to be added to account for idle times in the system, for there may be times when the queue(s) would be completely empty due to there being a gap between the execution of one job and the arrival time of the following job(s). Thus, I could not follow my original design of relying on the scheduling loop to run until the queue is empty, because it could be possible that queue is empty, yet there are still jobs that require to be run in the future due to the system currently being idle. Instead, I simply had a boolean variable representing whether all the jobs had finished running to completion to signify an end to the scheduler.

Throughout the process of working on this project, I of course learned how all these scheduling algorithms worked in detail, because having to write them out instruction by instruction forced me to get a deeper understanding about them, which will probably be more valuable than simply only reading about them in a book. Because of my deeper understanding gained, I was able to get the end result that I wanted and envisioned since the beginning. All of the output based on the provided test case and some other test cases worked perfectly as they should with and without idle times, and the graphs outputted correctly for all of these test cases. Thus, I am satisfied with the end result of this project.