Summary

Designing and implementing a functioning scheduler compatible with three of the main scheduling algorithms was an interesting endeavor, to say the least. Yet again, I found that there was a deep intricacy to something that would appear to be very simple. Finding the right sequence of coordination and steps between the different queues and incoming tasks proved to be more difficult than I had originally thought. It was a challenge that I was prepared to face with stride, however, because, of the projects that I had worked on this semester, it was the most in tune with my understanding right off the bat. Not to say that the understanding of the other projects did not come with time or through my implementation and research, but these scheduler algorithms came very naturally to me, both in learning, and in implementing.

From the very start, I decided that I would commit to writing the code neatly, and more importantly, as efficiently as possible. Nearing the end of the semester, the perfectionist in me surfaced and I took it as an opportunity to produce the best piece of code that I could. I had quickly noticed that there were many similarities in the process of setting up the schedulers that could be combined under a single abstract scheduler class. This prompted me to explore the idea of applying inheritance to my project. With a bit of research, I was able to design this abstract class such that it would process the incoming file and extract the relevant task information from it. Initially, I would read the inputs from the file line by line within the actual scheduling algorithm implementation, but I quickly found that it would prove to be redundant and suboptimal. It is then that I decided to implement a linked list style Job class that encapsulates the job information and chains together the tasks in the order of their arrivals to make it easy to traverse over the tasks in their chronological order. This proved to simplify my code in the upcoming algorithms greatly. From that point, I set up the remaining classes, which all inherited from the main scheduler class. The goal of these classes would be to override the virtual function “schedule” that would be called within their shared super constructor. It then came down to a bit of trial and error as I implemented round robin as best as I could. After a bit of debugging and thorough review, I was able to get the RR scheduler working as intended. It was nothing more than finding a way to properly manage the insertion times into the queue. For the shortest remaining time algorithm, I found that it was almost exactly the same as my RR implementation except that it used a priority queue, and made a few changes here and there. With the help of my trusty sidekick copy and paste, I found myself knocking on the doors of the feedback scheduler class. After my coordination troubles of round robin and learning from my mistakes, the feedback algorithm fell into place very quickly. I set up three queues within an array list and iterated through them, as long as they were not all empty, looking for the highest priority task to schedule. Honestly, after figuring out the setup and the RR implementation, the rest was smooth sailing.

I would say that overall, this project was probably quite a bit easier than the earlier ones. Whether that is due to my increased understanding initially or an objective lower complexity, I was happy with how quickly my brain figured out how to solve this one. I feel as though this one really solidified my understanding of a technique that would not be language specific, nor would it be niche and questionably useful, but instead something that I can carry with me in my normal daily life. I may quickly become known as the person who applies the feedback algorithm with exponentiation of quanta to his daily study routines and I find that hilarious. Thank you for a great semester yet again professor!