Amogh Yatnatti

Professor Ozbirn

CS 4348.001

1 May 2021

**Project 3 Summary**

I approached this project by breaking it down into a structure of pieces. I first broke it down into three blocks, one for each scheduler. I then looked at the first block or the FCFS section. For this section, I needed to know how the “jobs.txt” file would be structured. The instructions noted that this file would have the jobs sequentially ordered in terms of letters and arrival times. This meant that I could read in each job from the text file, append it into an array, and it would already be ordered. This made the FCFS scheduler very easy to implement. I simply had to traverse the array and print out each job’s schedule. I also made sure that we could accommodate any empty gaps between jobs for this scheduler. For example, if we have job A with an arrival time of 0 and a service time 1 and job B with an arrival time of 5, then we would have a gap where no job is running from 1 to 5-time units. I accounted for this in all schedulers by having a case where I check if the time is less than all non-completed jobs. If this happens, I change the time to the closest, upcoming job arrival time. With the time accommodation and the traversal working, I finished the FCFS scheduler.

I then moved onto the next block which was the SPN scheduler. This scheduler was significantly more difficult to implement compared to the FCFS one. I learned that I would need to keep track of numerous variables for each job, so I made a static class called Job to help organize them. In addition to each job’s name, service, and arrival time, I made sure to track the amount of whitespace needed for each output, the job’s index in the array, and if the job was completed. I had to keep track of these extra variables because the SPN scheduler can run the jobs in a different order. I also had to make a “waiting” array called that would keep track of all the jobs that had arrived and were not completed. I needed this array because I had to look at all of these jobs and select the shortest process to be run next. Once I found the shortest process, I would “run” it, but before I would mimic running it, I stored the number of spaces (equal to time before execution) I needed for the output. I then added its service time to the overall time and marked it completed. I then cleared the waiting array and looked for jobs waiting now (with the updated time). I did this until there were no more jobs waiting (waiting array is empty). To print the output, I traversed the ordered list of jobs, printed out the number of spaces I had assigned, and I printed an ‘X’ for every unit of service time. During each iteration, I also reset the space and completed variable so I could reuse them for the HRRN section.

After this, I moved onto the HRRN scheduler. I quickly learned that this approach is very similar to the SPN scheduler. It was interesting to see how similar these approaches were. Most of the code I used for the SPN could be reused for the HRRN section. The main difference between SPN and HRRN is how the next process is selected. For SPN, we choose the shortest process to go next. For HRRN, we choose the process with the highest response ratio to go next. After that, the process is exactly the same. Since the code was essentially the same, I decided to make them methods to reduce boiler-plating. I made one method that would get all the jobs that are waiting (getWaitingJobs). I made another method that would “run” the job and update the corresponding variables (runJob). I then made a method that would print the chart (printChart). With all of these methods, I was able to run both schedulers easily and significantly reduce the reused code I had.

The purpose of this project was to learn and mimic how a FCFS, SPN, and HRRN scheduler works. I was able to get matching outputs to the sample case, but I also tried to accommodate for edge cases as well. I made sure to accommodate for large time gaps between jobs and if multiple jobs arrive at the same time. The end result is that I understand how these schedulers work on a deeper scale than I had before. While I did not necessarily implement these concepts at a fundamental level, my simple replica gave me insight into what might be required to actually implement them.