Julio Berina

Dr. Young

CS 4310

27 Jun 2019

CS 4310 Project Report

The purpose of this programming project was to examine the average turn around time of four job scheduling algorithms learned. Technically, it's only three, but two of the algorithms tested are Round-Robin with two different time quantums. The details of which job is currently running and at what time it completes is to be displayed in either a Schedule Table or Gantt Chart format. To keep things simple, I made necessary assumptions while creating the program. I assumed that all jobs arrived at time 0 for testing Shortest Job First (SJF) and processes arrived in the order they were read from the file for the others.

For the assignment, I decided to with Java as my programming language of choice for implementing the algorithms. I kept the code looking as simple as possible to the point where only a few comments would be necessary. The rest of the code would be extremely straightforward and readable which makes commenting less needed. As far as my hardware tools, I did not use any special hardware for my project other than my own computer. I used Java, a text editor, and the terminal to compile and run the program.

When it came to displaying which job was being executed as well as the start time, end time, and job description, I decided to go with a Schedule Table instead of a Gantt Chart. The reason I went with a Schedule Table as opposed to a Gantt Chart was that it looked much better in my opinion for program output purposes. A Gantt Chart looks better when doing the

algorithms on paper, but a Schedule Table lends itself well to a terminal as program output. The Schedule Table is more descriptive of what's going on and grows vertically, which is how it looks better on terminal. A Gantt Chart grows horizontally, which wouldn't look good for the terminal and is a bit less descriptive and harder to read.

Besides the way I structured my code, the strength of my program lies in the concise and efficient implementation of each algorithm per execution. Each algorithm is separated into a method and implemented with as little code as possible. On top of keeping the methods short, things like variables were named appropriately to minimize commenting. The most important of implementing each algorithm was their efficient implementations. A hash table was used to store jobs, which provided constant time insertion and retrieval and linear time complexity (O(n log n) for SJF) per algorithm.

Although my program was implemented as concisely and efficiently as possible, there are flaws to it. One of the biggest problems with the program is that it expects the file it reads from to have a certain structure and order. It expects a job name followed by a number in the next line, and the program would have problems if the structure of the file deviated from this. For example, the program would have problems if the next job was two or more newlines away or if the job name and the burst time were on the same line. Another weakness of the program is that the algorithms were designed to make assumptions like AT = 0 for every job, which makes the program too domain-specific incapable of working with different arrival times.

If I were to simulate the entire operating system, I would start by renaming the Main class to Scheduler. I'd create a Scheduler class with the methods for each algorithm tweaked to be able to deal with different arrival times. On top of that, I would implement the preemptive

versions of those algorithms, if any, and simulate the OS in other parts of the project to provide different arrival times for jobs. I would have a Main class with a main method to execute the program and create classes for other components of the operating system. My approach is to build classes like OSProcess, OSThread, OSKernel, etc. as operating system components and have them interact with each other in a way that acts like an operating system.

In conclusion, the performance of each algorithm was pretty consistent per file. Shortest Job First (SJF) always displayed the fastest average turn around time at all times. First Come First Serve was the second fastest followed by Round Robin TQ = 2 (RR-2) and Round Robin TQ = 5 (RR - 5) being the slowest. For each file, RR-2 had an average turn around time of only < 3 time units less than RR-5, so it wasn't that much faster. RR-2 had more rows on its Schedule Table, but its smaller TQ made it faster than RR-5 with less rows but longer response time.