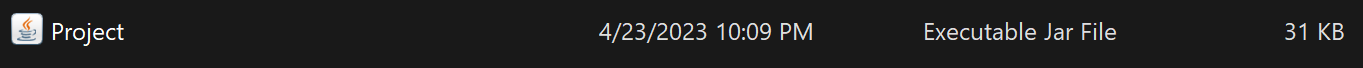
Tyler Marois

Operating Systems

Project 4

Running the Project:

Simply double click on the Project.jar file which will immediately launch my program, no need for compiling!



If you would like to still compile my code to be sure it actually works or if the Project.jar file doesn’t work then move into the Project\_Code folder where all my java code is stored. Once all the files have been compiled then run the Menu.java file with the command: java Menu.

Using the Program:

Upon Launch you will be brought to my menu three buttons will be displayed here:

1. Process Watcher – This was my idea of a fun interactive screen that shows the both schedulers running at the same time. I achieved this by running both of them in separate threads. Before you enter the Process Watcher you’ll be asked to input two integers the first being size, which is the how many processes in the test and second being the speed of the scheduler. The higher number you put for speed the slower it will go because it correlates to how long each unit T represents. If I didn’t add this modifiable delay the process would happen way too fast. Once both are done you can see the results by pressing the results button. There unfortunately is a bug with the results where they aren’t correct unless you use size 1000 ☹
2. Raw Results – Displays all of the data from section 3 part 2 of the project where we collect times of FCFS, and Round Robin with modifiers for context time and time slice.
3. Group Results – Displays the data from section 3 part 3 where we collect data based off groups ordered by normalized turnaround time.

How I Implemented:

Firstly, I figured I needed an object that could hold data based on the processes that would go through the schedulers, this led me to create the process class which stores all the data I’d need about the processes being “ran.”

Then, I started with FCFS being the easier of the two because we just let each process run till it finishes and they are sorted by arrival time. I created the scheduler method which simply removes the first thing in the processes list and simply decrements it’s service time till it hits 0, upon which I would add it to the Complete Processes list all the while incrementing T to keep track of time. I simply did this until completed processes reached the size that processes initially was. So FCFS was done quite simply.

Finally, the harder part was Round Robin. I had to start with implementing the min heap which I did with nodes that contain a process as a field. Once I had the min heap working I followed a very similar Idea from FCFS where I would remove from the min heap, but now I would decrease its service time based off the time slice and add it back to the heap if it wasn’t finished yet, otherwise it would go to completed processes. After a process got through it’s time slice I would add context switch to T if there was any context switch.

How the Data was Collected:

I made this process very simple by having all of the things I would need data on be fields within the Process class. Doing it this way allowed me to simply loop through any list of completed processes and extract any data I could possibly need. Most notably the turnaround time and normalized turnaround time. Each of the scheduler classes have methods that calculate averages of this data and then in the driver class I put it all together to make the program do all the data collection for me.

The Data:

Note that while the data was collected in milliseconds I have converted it all to seconds in all the data I show.

|  |  |  |  |
| --- | --- | --- | --- |
| Round Robin Individual Data | | | |
| Context Switch | Turnaround Time | Normalized Time | Time Slice |
| 0 | 124.5334 | 1.281 | 50 |
| 5 | 136.0871 | 1.3785 | 50 |
| 10 | 148.2967 | 1.4801 | 50 |
| 15 | 160.4692 | 1.6049 | 50 |
| 20 | 171.8797 | 1.6923 | 50 |
| 25 | 184.0822 | 1.8094 | 50 |
|  |  |  |  |
| 0 | 131.7472 | 1.2952 | 100 |
| 5 | 138.1808 | 1.3529 | 100 |
| 10 | 144.7309 | 1.4093 | 100 |
| 15 | 151.3291 | 1.4669 | 100 |
| 20 | 158.1865 | 1.53 | 100 |
| 25 | 164.4437 | 1.5872 | 100 |
|  |  |  |  |
| 0 | 167.0447 | 1.6405 | 250 |
| 5 | 170.6115 | 1.6608 | 250 |
| 10 | 174.2552 | 1.7123 | 250 |
| 15 | 177.3169 | 1.7228 | 250 |
| 20 | 180.8969 | 1.7626 | 250 |
| 25 | 184.2001 | 1.8073 | 250 |
|  |  |  |  |
| 0 | 246.2553 | 2.5386 | 500 |
| 5 | 248.7619 | 2.5611 | 500 |
| 10 | 251.2772 | 2.5887 | 500 |
| 15 | 253.7884 | 2.625 | 500 |
| 20 | 256.299 | 2.659 | 500 |
| 25 | 258.8149 | 2.6593 | 500 |

|  |  |
| --- | --- |
| FCFS Individual Data | |
| Turnaround Time | Normalized Time |
| 120.9108 | 1.639 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Group Data | | | | |
| Process Group | FCFS | RR 50 Time Slice | RR 250 Time Slice | RR 500 Time Slice |
| 1 | 0.0597 | 0.1202 | 0.1006 | 0.2456 |
| 2 | 0.18 | 0.3061 | 0.2957 | 0.6241 |
| 3 | 0.3003 | 0.4935 | 0.5092 | 0.9617 |
| 4 | 0.4212 | 0.6756 | 0.7265 | 1.2622 |
| 5 | 0.5423 | 0.8439 | 0.9339 | 1.5299 |
| 6 | 0.663 | 0.9941 | 1.1312 | 1.7687 |
| 7 | 0.783 | 1.1225 | 1.3104 | 1.986 |
| 8 | 0.9032 | 1.2313 | 1.4725 | 2.1883 |
| 9 | 1.0237 | 1.3256 | 1.6077 | 2.3429 |
| 10 | 1.1438 | 1.4065 | 1.7115 | 2.4375 |
| 11 | 1.269 | 1.4755 | 1.7903 | 2.4872 |
| 12 | 1.418 | 1.5352 | 1.85 | 2.5136 |
| 13 | 1.6076 | 1.5954 | 1.8933 | 2.5328 |
| 14 | 1.8557 | 1.6921 | 1.9355 | 2.5809 |
| 15 | 2.1944 | 1.9162 | 2.0333 | 2.8602 |
| 16 | 2.687 | 2.3512 | 2.3028 | 3.5993 |
| 17 | 3.4661 | 3.0482 | 2.9172 | 4.8093 |
| 18 | 4.8722 | 4.2251 | 4.2376 | 6.9641 |
| 19 | 8.2841 | 6.753 | 7.6229 | 12.1601 |
| 20 | 48.5444 | 41.2324 | 48.6356 | 73.8706 |

The Charts: Chart, bar chart

Description automatically generated

Chart, bar chart

Description automatically generated

I decided to only show up to 8 normalized turnaround time here since, The extreme high values of up to 40-70 of the longest process groups. It made the chart difficult to understand and there was only one definite loser in those ranges that being RR with a 500ms time slice.

Conclusion:

I believe overall Round Robin wins this contest, due to it’s ability to handle a wider range and produce low turnaround times for larger processes. Although, a case could definitely be made that since FCFS produced better with the smaller processes that it is better in that regard, but if we are looking for something to handle all sorts of processes Round Robin is definitely the way to go.

The Round Robin that performed the absolute best was with 0 context switch and 50 time slice, but I don’t think that’s enough for me to declare it the winner. Firstly assuming a context switch of 0 is hard to say, and secondly that is the only context that the 50 time slice Round Robin won in. This is why I’m choosing the 100ms time slice Round Robin as my winner since it performed best in normalized times in 5 out of the 6 context times. This clearly showed me that 100ms time slice has more adaptability and can handle any context time better than any other time slice could.

Final Notes:

* I Tried a lot of new things in this project if anything breaks please let me know!
* I included some javadoc’s with this project since I felt there was quite a bit going on And it should be well documented, you can find it here: [Javadoc](https://turing.plymouth.edu/~tm1207/WebFiles/TMarois_OS4310_Proj4/docs/package-summary.html)
* The Process Viewer may not always be an actual perfect representation of what’s going on since I had to modify some things around to get it show correctly and there’s still that error in the results unless the size is 1000 (which I suggest trying with speed 1 if you have time, you’ll see the same results I obtained). Either way I still find it fun to see the time tick down and watch them move the processes in and out. Also the Round Robin part is using context time 10 and time slice 50.
* Threads were a huge help in this project, but also a huge hassle but I really enjoy how it all turned out. I wish wait() worked more like it did in c.