California State Polytechnic University, Pomona

CS431 Operating Systems

Dominick Atanasio

Project 1

OS Process Scheduling Simulations

10.13.2017

Jacob Lepere

012839151

**Introduction**

The purpose of this project is to implement different process scheduling algorithms and analyze their performances. The four scheduling algorithms implemented are First-Come-First-Serve, Shortest-Job-First, Round-Robin, and Lottery. First-Come-First-Serve and Shortest-Job-First run each process on the CPU until completion before scheduling a new process while Round-Robin and Lottery run each process on the CPU for a specified time quantum before adding it back into the queue, if unfinished, and then choosing a new process to run. The Round-Robin algorithm is simulated twice with time quantum 25 and 50 while the Lottery algorithm is simulated once with time quantum 60. It is also important to note that three units of CPU time is lost while switching processing.

Compiling and running the program is defined in the README.txt file associated with the project submission folder. For this reason, I will skip these details, but it is important to note the format of the output file. The format of the output file is a csv containing header information for the run, such as input filename and system time, followed by data for each implemented scheduling algorithm. Data for each scheduling algorithm contains the following: the name of the algorithm, a matrix for the raw data of the simulation, and the turnaround time for each process. The total and average turnaround time is included as well.

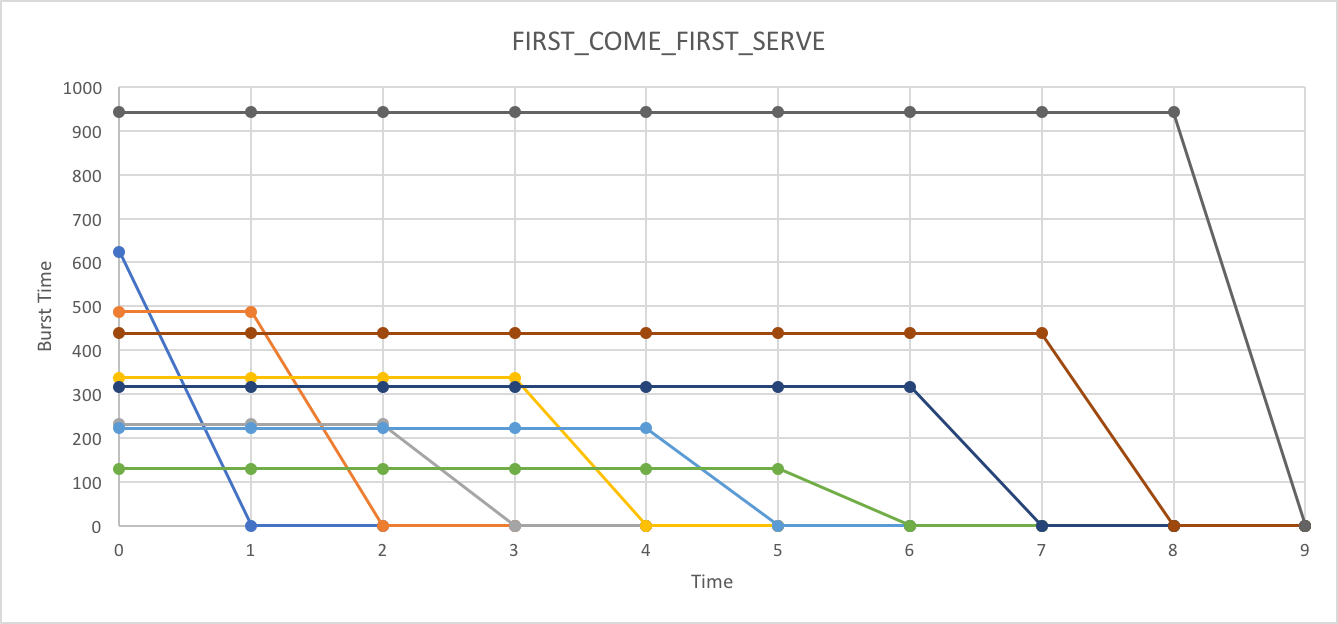
**Analysis**

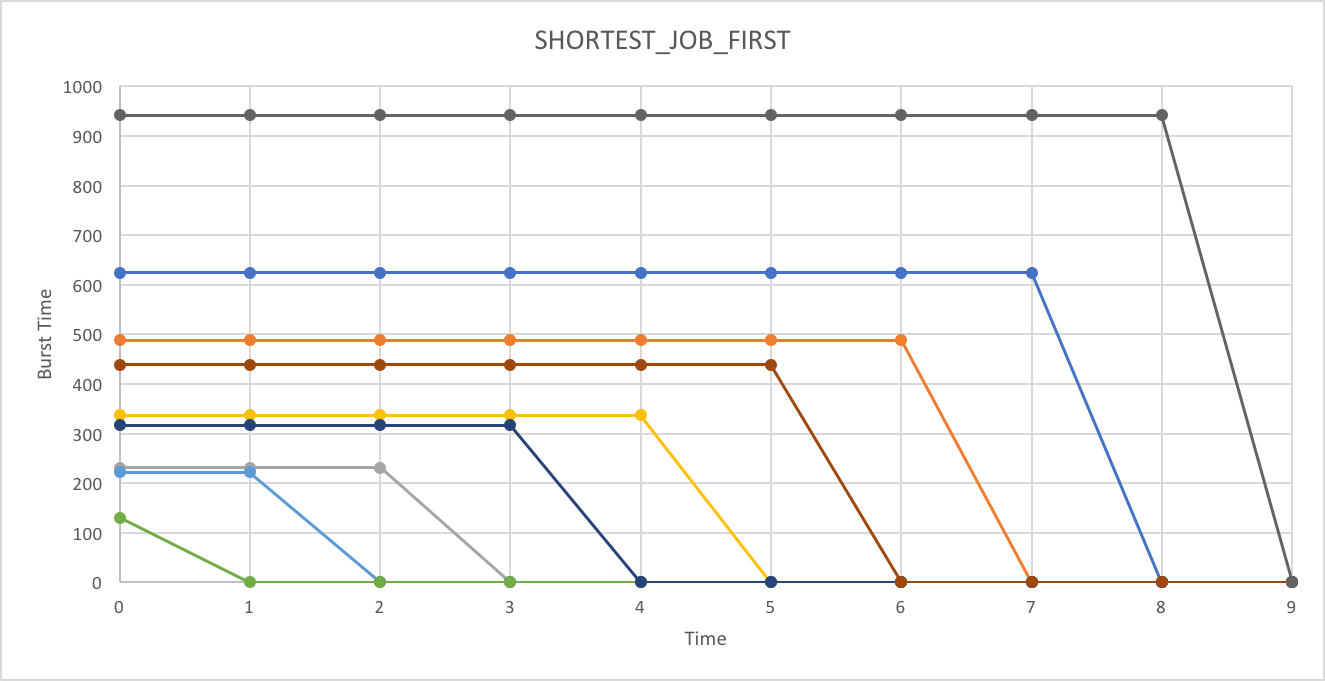
The data matrix for each scheduling algorithm was chosen to maximize analysis. The columns for the matrices are the processes while the rows are defined by a counter which is incremented each time a process is swapped on the CPU. That is, time 0 is when no processes have run on the CPU, time 1 is after the first process has run on the CPU for some time and before the next process has run on the CPU, time 2 is after the second process has run on the CPU for some time and before the next process has run, and so forth. Each data entry is the remaining burst time for the corresponding process and CPU swap count. See below for an example of the Shortest-Job-First algorithm. (All data presented in this report is from the testdata1.txt input file present in the submission folder.)

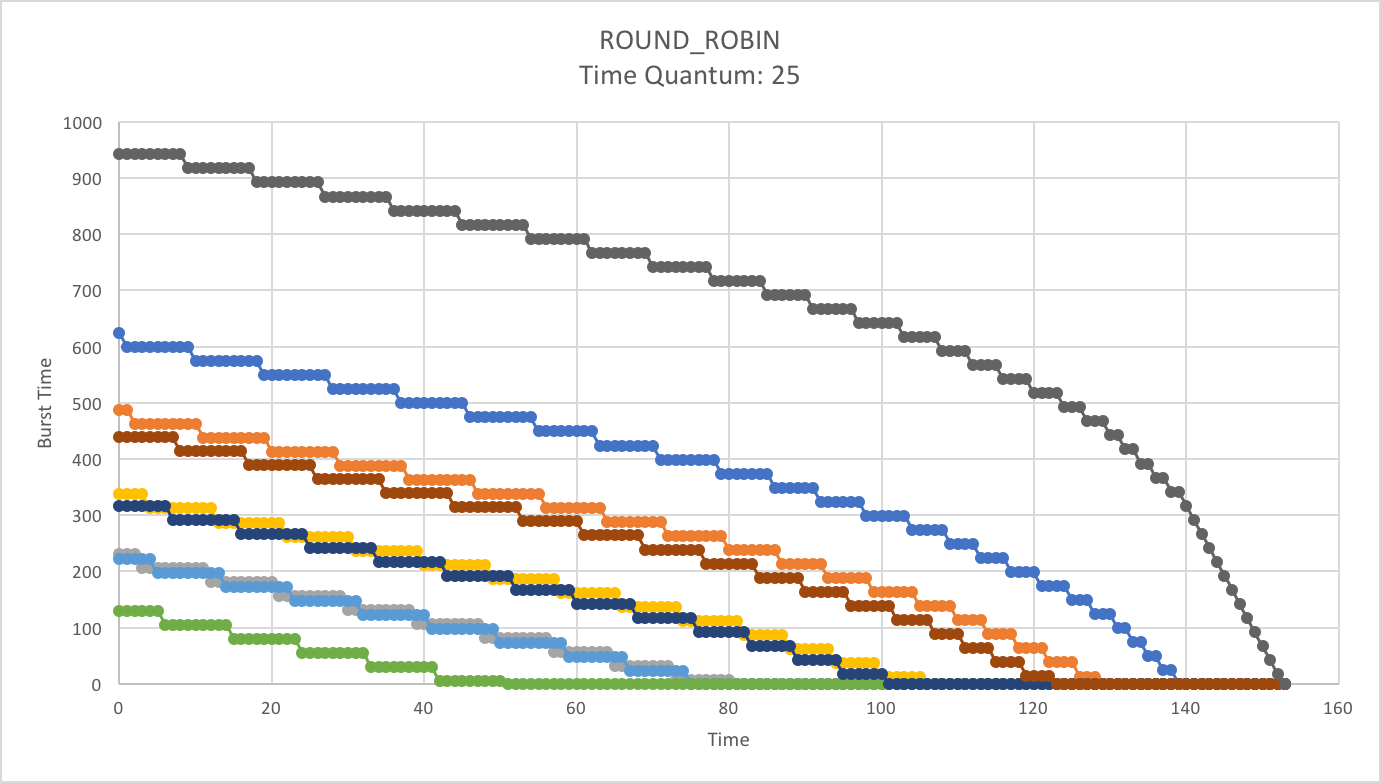
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PID: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| BURST\_TIME: | 624 | 488 | 231 | 337 | 222 | 130 | 317 | 439 | 942 |
| PRIORITY: | 2 | 15 | 16 | 7 | 3 | 17 | 11 | 8 | 10 |
| TIME | PID\_1 | PID\_2 | PID\_3 | PID\_4 | PID\_5 | PID\_6 | PID\_7 | PID\_8 | PID\_9 |
| 0 | 624 | 488 | 231 | 337 | 222 | 130 | 317 | 439 | 942 |
| 1 | 624 | 488 | 231 | 337 | 222 | 0 | 317 | 439 | 942 |
| 2 | 624 | 488 | 231 | 337 | 0 | 0 | 317 | 439 | 942 |
| … | … | … | … | … | … | … | … | … | … |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 942 |
| 9 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

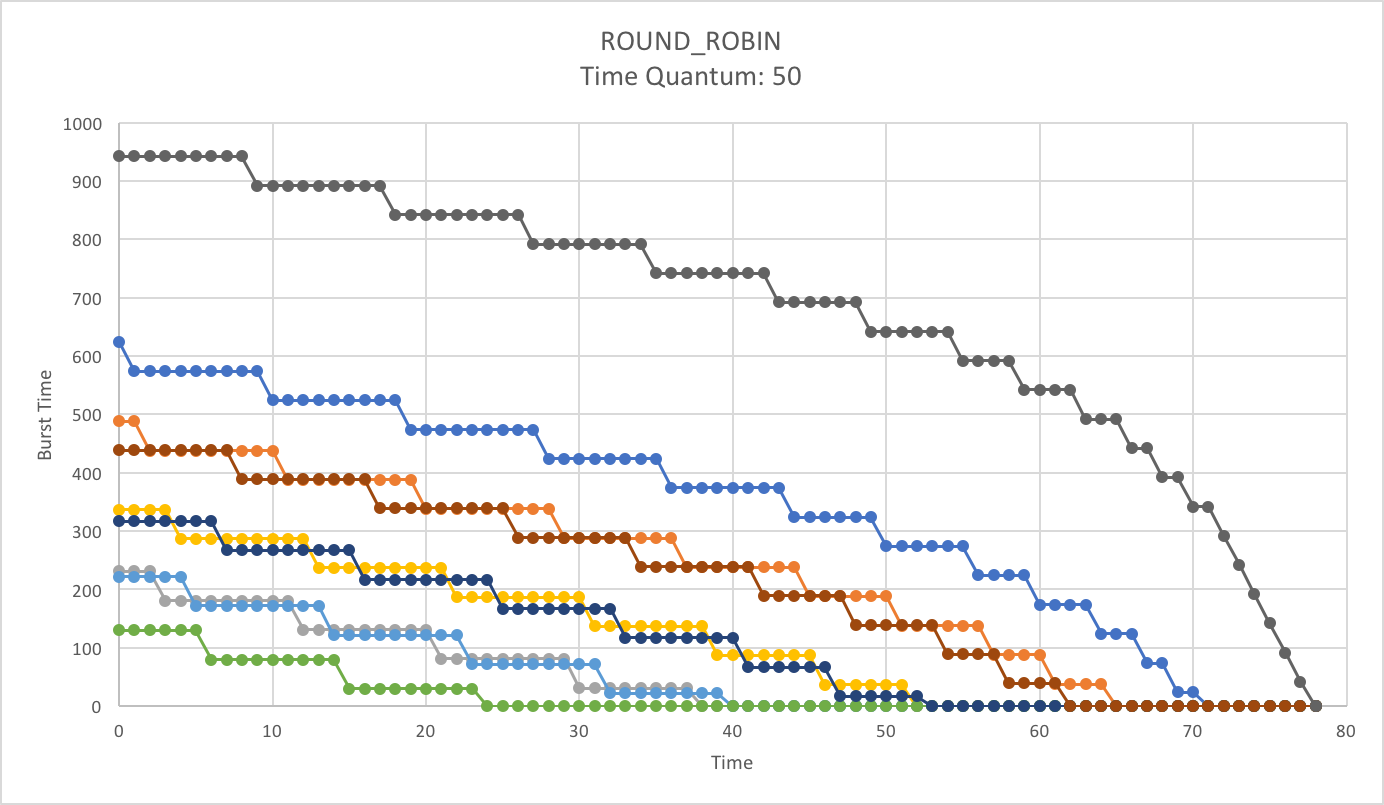
A graph is recommended to analyze this data as it presents a visual representation for the transitions between processes running on the CPU as well as how burst times and priorities may influence the frequency at which a process is scheduled. Following is a graph of the data for each scheduling algorithm.

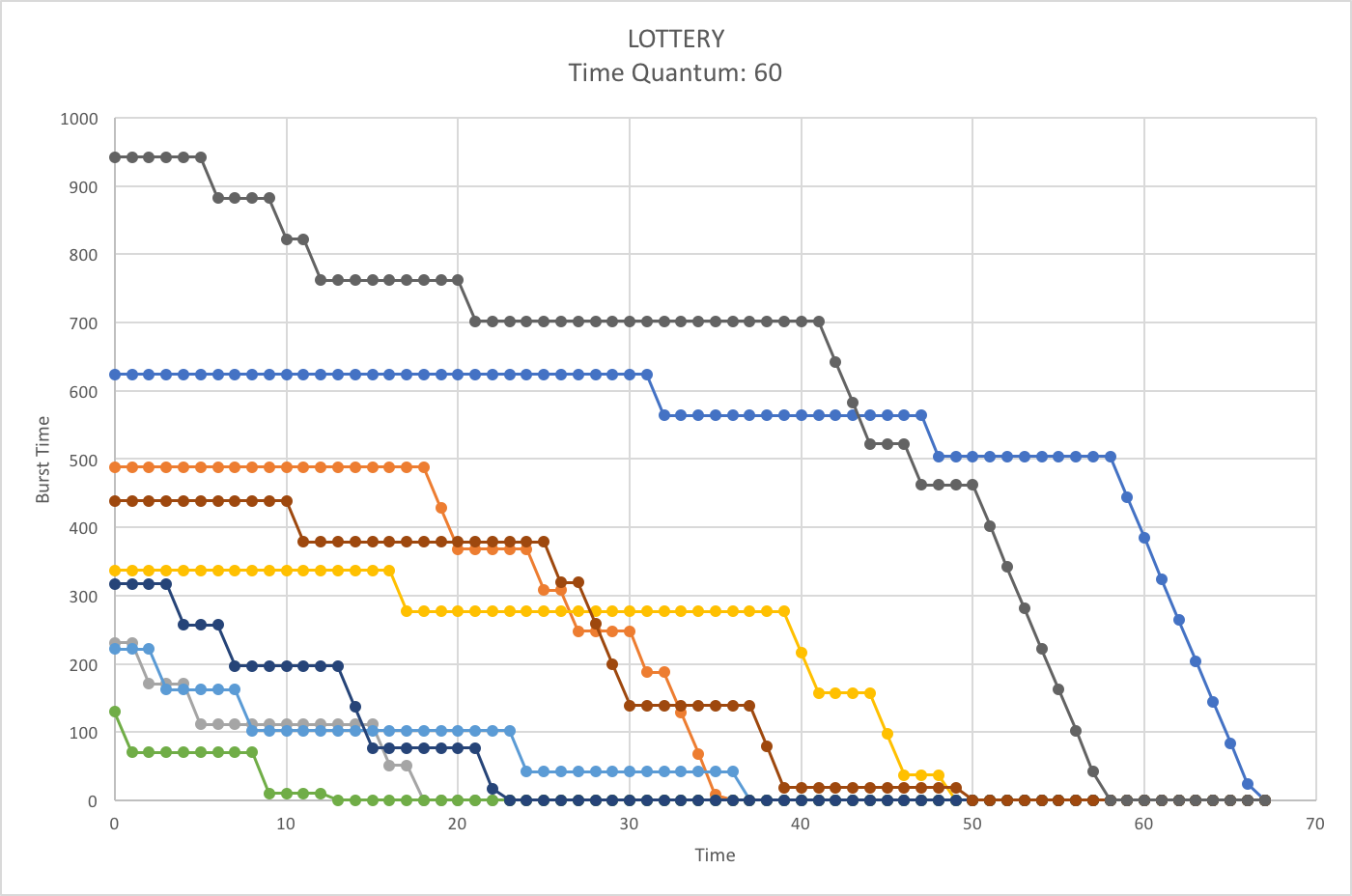
../../../../../../Screen%20Shot%202017-10-11%20at%203.46.02%20PM











Another area for analysis relevant to the performance of a scheduling algorithm is the total and average turnaround time. This data, along with the turnaround time for each process, is shown below.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| TURN\_AROUND\_TIME | | | | | | | | | | | |
| SCHEDULER | PROCESS\_ID | | | | | | | | |  |  |
| PID\_1 | PID\_2 | PID\_3 | PID\_4 | PID\_5 | PID\_6 | PID\_7 | PID\_8 | PID\_9 | TOTAL | AVERAGE |
| FIRST\_COME\_FIRST\_SERVE | 624 | 1115 | 1349 | 1689 | 1914 | 2047 | 2367 | 2809 | 3754 | 17668 | 1963.1 |
| SHORTEST\_JOB\_FIRST | 2809 | 2182 | 589 | 1249 | 355 | 130 | 909 | 1691 | 3754 | 13668 | 1518.7 |
| ROUND\_ROBIN\_(25) | 3802 | 3523 | 2223 | 2902 | 2074 | 1405 | 2775 | 3367 | 4186 | 26257 | 2917.4 |
| ROUND\_ROBIN\_(50) | 3598 | 3306 | 1972 | 2673 | 2050 | 1249 | 2693 | 3159 | 3961 | 24661 | 2740.1 |
| LOTTERY\_(60) | 3928 | 2111 | 1072 | 2889 | 2156 | 766 | 1344 | 2911 | 3397 | 20574 | 2286.0 |

Although the graphical representation of the transitions between processes suffices to indentify when and for how long a process has run, making use of Excel functions can easily output this data. To do this, use the Excel functions below on the column to the right of the matrices.

‘=CONCATENATE(IF(B11<>B10,$B$9,""),IF(C11<>C10,$C$9,""),IF(D11<>D10,$D$9,""),…)’

‘=SUM(IF(B11<>B10,B10-B11,0),IF(C11<>C10,C10-C11,0),IF(D11<>D10,D10-D11,0),…)’

This can output the process id or the time ran for the process that was just run on the CPU.

**Conclusion**

The four algorithms chosen for the project give an idea of the different algorithms implemented for different systems. On a batch system where one of the main goals is to maximize throughout, First-Come-First-Serve and Shortest-Job-First are acceptable algorithms. Interactive systems, on the other hand, need to please the average user who does not understand nor care why their computer can’t run multiple programs at once. Round-Robin and Lottery are better choices in this environment because they constantly switch between processes.

The performance of these two classes of algorithms can be identified by their respective throughputs. Algorithms for batch systems perform the least amount of process swaps to minimize this added cost on the throughput. Algorithms for interactive systems, on the other hand, need to constantly swap processes, increasing the throughput. For our example data, First-Come-First-Serve and Shortest-Job-First perform 8 process swaps while 125, 77 and 66 process swaps occur for Round-Robin 25-time quantum, Round-Robin 50-time quantum and Lottery 60-time quantum, respectively. Although this seems cumbersome, with a CPU swap time of only 3 time units, only a small fraction of turnaround time is devoted to switching processes. For Round-Robin 25-time quantum, only 375 units of time is devoted to switching processes, which is only 1.43% of the total turnaround time. Although this is a simulation with a provided CPU swap time, this data shows that a lower time quantum might be beneficial in an interactive system because the total overhead from CPU swaps may only be a small fraction of the total turnaround time.

I am hesitant to analyze the lottery algorithm because the time to choose a process has not been taken into consideration. Summing up the burst time of all processes, generating a random number, and finding which process to run takes time, but it is unclear exactly how much time is needed in the simulation. With 66 process switches, for our example, this could generate a bottleneck on the scheduler. Also, there exists a slightly negative correlation, -0.274, between burst time and priority for the data set. That is, the lower burst times have higher priorities. Therefore, the lower burst times are more likely to run first, leading to a lower turnaround time. Burst time and priority should be independent, or at least they could be. It is possible that a negative correlation is desired so shorter processes are more likely to run, but this is not a requirement.

If I were to repeat the experiment, I would use more CPU swap times for the Round-Robin algorithm and I would limit or fluctuate the dependency between burst time and priority for the Lottery algorithm. This would provide more insight to draw conclusions regarding the performance of these algorithms.