Performance Report

By: Dylan Silva-Rivas

# Algorithm Overview

I will be comparing first-come first-serve (FCFS) and shortest-job first (SJF) CPU scheduling algorithms. FCFS takes in the very first process that arrives to the CPU and runs until its burst time is reaches it limit and repeats the process until there are no more processes left to run. SJF is the opposite, instead of taking the first process that arrives to the CPU, it observes the shortest burst time of each process and takes the smallest one and runs it.

# Methodology

Here is the approach to my methodology: General steps and detailed steps. Here are the general steps:

1. Create constant data in the main function to pass to other functions.
2. Create CPU scheduling function.
3. Randomize data.

Now for the detailed steps:

1. For the first general step, I created a Process struct. This struct contains the attributes process ID, arrival time, burst time, wait time, and turnaround time. Data will be sorted depending on what’s the most important. For example, if the program decides to use FCFS, arrival time will be prioritized when sorting. For SJF, burst time will be prioritized when sorting. The reason for sorting is so accessing specific processes faster depending on the scheduling algorithm. I will be using merge sort when sorting the processes, which is O(n\*log(n)).
2. After sorting data relative to the arrival time, the data will be thrown into the CPU scheduling function to calculate waiting time, turn around time, and idle time.
3. After sorting data relative to the burst time, the data will be thrown into the CPU scheduling function to calculate waiting time, turn around time, and idle time.
4. Random data to measure performance over time for 100 iterations.

# Performance Metrics

The performance metrics that I am required to measure is wait time, turnaround time, and idle time. The way I approached this, as mentioned earlier, is with random data. I felt like feeding these algorithms random data would be the most efficient way to measure their differences. There is a loop that runs 100 times. It generates a random amount of processes, the processes will be assigned random burst time and arrival time. After that, the data will be sorted to compute FCFS and SJF. After the main loop is done running, calculations for all averages will occur.

# Comparison and Analysis

Since the data is random, I will show my data specifically. The output will be different, but the comparison and analysis will be the same. I will present 5 executions of the program.

* First execution:

Averages for 100 iterations of FCFS for 753 processes

Avg wait: 34.55

Avg T.T: 46.66

Avg idle: 6.37

Averages for 100 iterations of SJF for 753 processes

Avg wait: 31.43

Avg T.T: 43.54

Avg idle: 12.00

* Second execution:

Averages for 100 iterations of FCFS for 743 processes

Avg wait: 35.27

Avg T.T: 47.62

Avg idle: 6.66

Averages for 100 iterations of SJF for 743 processes

Avg wait: 32.25

Avg T.T: 44.60

Avg idle: 11.93

* Third execution:

Averages for 100 iterations of FCFS for 748 processes

Avg wait: 37.23

Avg T.T: 49.77

Avg idle: 6.38

Averages for 100 iterations of SJF for 748 processes

Avg wait: 33.51

Avg T.T: 46.05

Avg idle: 12.24

* Fourth execution:

Averages for 100 iterations of FCFS for 741 processes

Avg wait: 37.52

Avg T.T: 50.10

Avg idle: 6.77

Averages for 100 iterations of SJF for 741 processes

Avg wait: 34.49

Avg T.T: 47.06

Avg idle: 13.67

* Fifth execution:

Averages for 100 iterations of FCFS for 736 processes

Avg wait: 37.05

Avg T.T: 49.48

Avg idle: 6.80

Averages for 100 iterations of SJF for 736 processes

Avg wait: 34.20

Avg T.T: 46.63

Avg idle: 12.68

It seems like overall, avg wait time and avg turnaround time for SJF is slightly better than FCFS. But, average idle time of SJF seems to be nearly twice as long than the average time of FCFS. The Visual\_Data.xlsx file will contain a visual representation of the data.

# Gantt Charts

With the way I approached comparing the scheduling algorithms, I can’t present each iteration with a gantt chart, since one execution runs 100 iterations for each scheduling algorithm. But, I can take a single iteration and represent it as gantt chart:

* Data:

Total time to run all processes: 93

Total idle time: 7

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Pi | A.T | B.T | W.T | T.T |

| P3 | 7 | 18 | 0 | 18 |

| P1 | 12 | 15 | 13 | 28 |

| P5 | 13 | 8 | 27 | 35 |

| P0 | 14 | 5 | 34 | 39 |

| P2 | 14 | 14 | 39 | 53 |

| P6 | 14 | 10 | 53 | 63 |

| P4 | 18 | 16 | 59 | 75 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Total time to run all processes: 100

Total idle time: 14

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

| Pi | A.T | B.T | W.T | T.T |

| P0 | 14 | 5 | 0 | 5 |

| P5 | 13 | 8 | 6 | 14 |

| P6 | 14 | 10 | 13 | 23 |

| P2 | 14 | 14 | 23 | 37 |

| P1 | 12 | 15 | 39 | 54 |

| P4 | 18 | 16 | 48 | 64 |

| P3 | 7 | 18 | 75 | 93 |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Gantt charts will be provided in the Gantt\_Chart\_Data\_One\_Iteration.xlsx file.

# Pros and Cons

I will pinpoint the pros and cons for each CPU scheduling algorithm.

* Pros FCFS:
  + Simple implementation.
  + Useful for long running processes.
  + Guarantees that every process will run.
* Cons FCFS:
  + Long waiting times.
  + Starvation.
  + Lower CPU utilization.
* Pros SJF:
  + Simple implementation.
  + Shortest jobs are favored.
  + Optimal average turnaround time.
* Cons SJF:
  + Starvation.
  + Large overhead.
  + Requires knowledge of execution time.

# Conclusion

Overall, during my statistical analysis, it seems like SJF performs slightly better than FCFS. But, the most surprising discovery is that SJF and FCFS follow the same algorithmic instructions, they just differ because of two different variables that they prioritize: Arrival time and burst time. This requires some sorting before using the sorting algorithms to get output.