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 the specific data that I was given. The output will be different, but the comparison and analysis will generally be the same. I will present 5 executions of the program.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | FCFS | | | SJF | | |
| Iterations | Avg. Wait Time | Avg. Turnaround time | Avg. Idle Time | Avg. Wait Time | Avg. Turnaround time | Avg. Idle Time |
| 1(753 Processes) | 34.55 | 46.66 | 6.37 | 31.43 | 43.54 | 12 |
| 2(743 Processes) | 35.27 | 47.62 | 6.66 | 32.25 | 44.6 | 11.93 |
| 3(748 Processes) | 37.23 | 49.77 | 6.38 | 33.51 | 46.05 | 12.24 |
| 4(741 Processes) | 37.52 | 50.1 | 6.77 | 34.49 | 47.06 | 13.67 |
| 5(736 Processes) | 37.05 | 49.48 | 6.8 | 34.2 | 46.63 | 12.68 |

It seems like overall, average wait time and average 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. Here is the 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 use a single iteration and utilize that data.

Keys:

* A.T = Arrival time
* B.T = Burst time
* W.T = Wait time
* T.T = Turnaround time

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FCFS | | | | |
| 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: 93

Total idle time: 7

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

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| SJF | | | | |
| 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 |

Total time to run all processes: 100

Total idle time: 14

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

The tables translated to the gantt charts:

The gantt chart above shows the turnaround time, wait time, burst time, and arrival time of each process for the FCFS algorithm.

The gantt chart above shows the turnaround time, wait time, burst time, and arrival time of each process for the SJF algorithm.

# 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

As stated earlier, SJF performs slightly better than FCFS in terms of wait time, turn around time. But when it comes to idle time, FCFS performance is two times faster than SJF performance. Performance, overall isn’t out of this world, which means there aren’t many real-life applications of these algorithms. But these algorithms are one of the earliest to be developed, meaning that modern CPU scheduling algorithms most likely utilize them.