Name: **Joshua Ngui**
Date: **3/22/2025**
Class: **CSC 4320**
CRN: **17449**

Project 1 Report

Overview

For this project, I chose to implement two scheduling algorithms: “First-Come, First-Serve” (FCFS), where the first process that arrives runs first, and “Shortest Job First” (SJF), where the process with the smallest burst time runs first.

Implementation Details

Before doing any process scheduling, my program first obtains process info from a user-specified file and stores that info in an array of `Process` objects, containing each process’s PID, arrival time, burst time, and priority value. Both algorithms are implemented in the form of a `while` loop that breaks once all processes have finished running. Before the loop, an integer variable named `timer` that tracks the time since process scheduling began is initialized at 0, and at the end of each cycle in the loop, `timer` is incremented by 1.

At the start of each cycle, after confirming that there are still processes left to run, the program checks if there is a process currently running. If a process is running, and said process has been running for its specified burst time, that process is marked as complete, and a new process can run. After this check, if no process is currently running, the program begins the operation of determining a new program to run. This is done by first marking each process whose specified arrival time has passed and has not already been completed as ready to be run. The processes ready to run are then compared with each other to

determine which process should run next based on the algorithm being used. If FCFS is being used, the process with the earliest arrival time is selected to run next, while if SJF is used, the process with the shortest burst time is selected. If multiple processes are in contention, then the process with the highest priority value is selected to run.

After all processes are finished running, a text-based Gantt chart is displayed, showing each process in the order they were run in and the times that they began and ended. Below that, the waiting time (time between arrival and starting), turnaround time (time between arrival and completion), and CPU utilization (percentage of time between arrival and completion spent running) of each process is displayed, as well as the average waiting time and turnaround time of all processes.

Results

My tests for my program utilized the following `processes.txt` file:

```
PID Arrival_Time Burst_Time Priority
1   0            5          2
2   2            3          1
3   4            2          3
4   1            3          1
5   10           6          2
6   10           9          3
```

Using this data, the following Gantt chart was displayed for FCFS:

```
| P1 | P4 | P2 | P3 | P6 | P5 |
0    5    8    11   13   22   28
```

For SJF, the following Gantt chart was displayed:

Name: **Joshua Ngui**
Date: **3/22/2025**
Class: **CSC 4320**
CRN: **17449**

```
| P1 | P3 | P2 | P4 | P5 | P6 |
0    5    7    10   13   19   28
```

As is shown by these Gantt charts, both algorithms ended up taking the same amount of total time to finish running every process.

For FCFS, the average waiting and turnaround times for the processes were as follows:

```
Average Waiting Time: 5.3333335
Average Turnaround Time: 10.0
```

For SJF, the average times were as thus:

```
Average Waiting Time: 4.5
Average Turnaround Time: 9.166667
```

As is shown, both the average waiting time and turnaround time were shorter with SJF than with FCFS.

Challenges & Solutions

I encountered several difficulties during the process of programming. One difficulty I encountered was needing to add to and delete from arrays of different objects but not having that functionality by default in Java. I solved this problem by using the `java.util.ArrayList` class, which is a type of array that can be added to and deleted from. Another difficulty I encountered was saving the starting and ending times of processes when they started and finished running, respectively. I solved this problem by creating a new class called `ProcessInfo`, which contained this information for each process.