

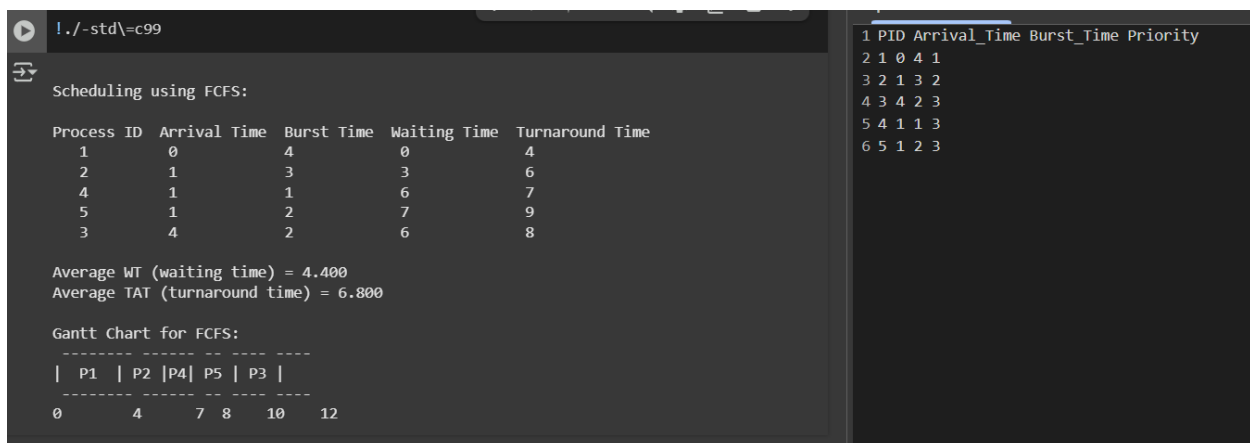
Github link: https://github.com/ceciliamuniz/OS_Project1

1) What scheduling algorithms I implemented

For this project, I chose to implement the First-Come, First-Serve (FCFS) scheduling algorithm and the Shortest Job First (SJF) scheduling algorithm. FCFS sorts the processes based on arrival time. The earlier a process arrives, the quicker it gets processed. SJF sorts the process based on the smallest burst time. However, for both algorithms, it's important to note the arrival time, as whichever process arrives at time 0, will start running until it's finished (both algorithms are non-preemptive, so I don't have to consider the process being interrupted by another process). As for the processing order when arrival time and burst time are the same for more than one process, here, FCFS chooses whichever process is read first, and SJF chooses based on burst time and then by priority.

2) Sample test cases and results

Sample test case for FCFS:



```
!./-std=c99

Scheduling using FCFS:

Process ID  Arrival Time  Burst Time  Waiting Time  Turnaround Time
1           0           4           0            4
2           1           3           3            6
4           1           1           6            7
5           1           2           7            9
3           4           2           6            8

Average WT (waiting time) = 4.400
Average TAT (turnaround time) = 6.800

Gantt Chart for FCFS:
-----
| P1 | P2 | P4 | P5 | P3 |
-----
0      4      7  8    10  12
```

PID	Arrival_Time	Burst_Time	Priority
1	0	4	1
2	1	3	2
3	4	2	3
4	1	1	3
5	1	2	3
6	5	1	3

For this test, 3 of my 5 processes share the same arrival time. Therefore, to determine which process runs first, my algorithm chooses which process came first (was read first) in the input file. If I was using a priority-based algorithm, the scenario would be different. Process 1 runs first as it arrives at time 0, then processes 2, 4, and 5, which were the ones that arrived at time 1, then process 3, which arrived at the latest at time 4. The average waiting time was 4.4 and the average turnaround time was 6.8.

Sample test case for SJF:

The screenshot shows a terminal window with the following output:

```

./-std=c99

Scheduling using SJF:

Process ID  Arrival Time  Burst Time  Waiting Time  Turnaround Time
1           0             4            0             4
4           1             1            3             4
5           1             2            4             6
2           1             3            6             9
3           4             2            6             8

Average WT (waiting time) for SJF = 3.800
Average TAT (turnaround time) for SJF = 6.200

Gantt Chart for SJF:
-----
| P1 | P4 | P5 | P2 | P3 |
-----
0       4  5  7     10  12

```

To the right of the terminal is a file named `processes.txt` with the following content:

```

1 PID Arrival_Time Burst_Time Priority
2 1 0 4 1
3 2 1 3 2
4 3 4 2 3
5 4 1 1 3
6 5 1 2 3

```

For this test, I chose to keep the same 5 processes I used for FCFS scheduling to show that arrival time doesn't matter as much for SJF scheduling. The results changed: process 1 still runs uninterrupted, as SJF is non-preemptive (a process can't be interrupted). While processes 2, 4, and 5 arrive simultaneously, process 4 has the least burst time (1), so it gets finished first, then process 5 (2), then process 2 (3). Process 3 arrived the latest, so it gets finished last. The average waiting time reduced from 4.4 to 3.8 and the average turnaround time went from 6.8 to 6.2. It's interesting that they both lowered 0.6 seconds.

3) Any challenges you faced

I faced many challenges while completing this project.

- 1) As a grad student, I had to do it all by myself. In a way it was good as I wanted to prove to myself that I could finish this project alone. I hope I did it right because the instructions confused me a bit.
- 2) I had to fix many pieces of code to make sure the input was coming from the processes.txt file in the correct format.
- 3) I wanted to use VSCode but I couldn't get it to compile C code.
- 4) I had to recreate the processes.txt file every time I went back to work on the project on Google Collab, because for some reason the file was always being deleted.
- 5) The calculations didn't work properly many times.
- 6) The file wasn't read correctly many times.
- 7) I had to learn how to use github in a short period of time, though I had it easy as I wasn't collaborating with anyone.