

Jefferson Kline

011734323

CPTS 360 programming assignment 2

10/24/25

My program's link is: <https://github.com/CPTS360FA25/pa-2-JeffLepp.git>

This programming assignment focused on implementing three scheduling algorithms: First Come First Serve (FCFS), Shortest Job First (SJF), and Round Robin(RR). The program reads data on what processes are benign given, then it simulates CPU and I/O executions for each cycle needed. It also reports details on Ticks spent waiting, IO ticks used, Arrival time, utilization statistics, turnaround, etc. I verified this by finding the expected outputs and comparing them with the outputs given in the code.

The general flow of the project goes as follows,

1. Read input file and populate process list
2. Sort process list based on arrival time and process ID
3. For each scheduling algorithm (FCFS, SJF, RR):
  4. Reset process list to initial state
  5. Simulate the scheduling algorithm
  6. Print process specifics and summary data

As given, random numbers are read from a file “random-numbers” which generates CPU Burst lengths using randomOS().

### **Implementation of Algs:**

**FCFS:** The FCFS algorithm I made schedules processes by their arrival time, and is very simple. When a process adopts the READY state, it is added to the FIFO (Ready) queue. Whatever process is at the front, the CPU will run until completion or an I/O burst. When there are **ties** in arrival time, we simply enqueue them in ascending order of their process ID's. When the I/O burst completes, the process enters the Ready queue.

**SJF:** The SJF algorithm I made schedules processes via their remaining CPU time, assuming the process is ready of course. I made pick\_next\_sjf() to help it scan all the READY processes for each cycle, which aids in finding the process (best) with the lowest CPU time needed to complete. If there is a **tie**, we break it by choosing the process with the smallest arrival time. If those are tied as well, we go with the smallest process ID.

**RR:** The RR algorithm I made uses a fixed quantum of 2 cycles. When processes enter the Ready queue they each receive 2 CPU ticks (cycles) before they either finish, get blocked for I/O, or being requeued at the back of the queue (FIFO). I use the same **tie** breaking rules from FCFS for when processes arrive at the same time.

I handle the tie breaking with my `tie_wrapper()` function, which prioritizes lower arrival times and small process ID's if needed.

#### **Data structures and functions:**

Each process is represented by a struct data type which includes: arrival time (A), CPU burst limit (B), total CPU time (C), multiplier (M), process ID, and runtime states.

A Queue made for processes was made in FIFO format.

We include a couple of global counters (`CURRENT_CYCLE`, `CPU_BUSY_TICKS`, etc.)

#### **Information sources that helped me:**

Helped me with understanding the FCFS and SJF and collectively the basis for all of the scheduling algorithms, since the main difficulty came with doing the first one, I made sure to interpret it and develop my own method from it!

<https://www.geeksforgeeks.org/operating-systems/program-for-fcfs-cpu-scheduling-set-1/>

<https://www.geeksforgeeks.org/operating-systems/program-for-round-robin-scheduling-for-the-same-arrival-time/>