

**The Superior University**

**Project Title**

**GHANTT CHART GENERATOR IN SCEDULING USING FCFS**

**👥 Group Members**

* MUSAB 050
* HAMZA 017
* UMER AMJAD 040
* MOIZ
* TAIMOOR

**📂**

**GitHub Repository**

**GitHub Repository Link:**

**https://github.com/musab-18/gantt-chart-generator-fcfs.git**

**🔧 Scheduling Algorithm Implemented**

* ✅ FCFS (First Come First Serve)
* SJF (Shortest Job First – Non-Preemptive)
* SJF (Preemptive)
* Round Robin

**📄**

**Project Description**

### **Problem Solved**

This project helps visualize how the **First-Come, First-Served (FCFS)** CPU scheduling algorithm works by simulating process execution and showing how each process is scheduled over time. It makes it easier for students and developers to understand scheduling behavior and performance metrics in operating systems.

### Required Inputs

**Arrival Time**: When each process arrives in the queue.

**Burst Time**: The CPU execution time required by each process.  
(Note: Time Quantum is not used in FCFS; it's used in Round Robin.)

### Generated Outputs

**Completion Time** of each process

**Turnaround Time** (TAT = Completion Time - Arrival Time)

**Waiting Time** (WT = Turnaround Time - Burst Time)

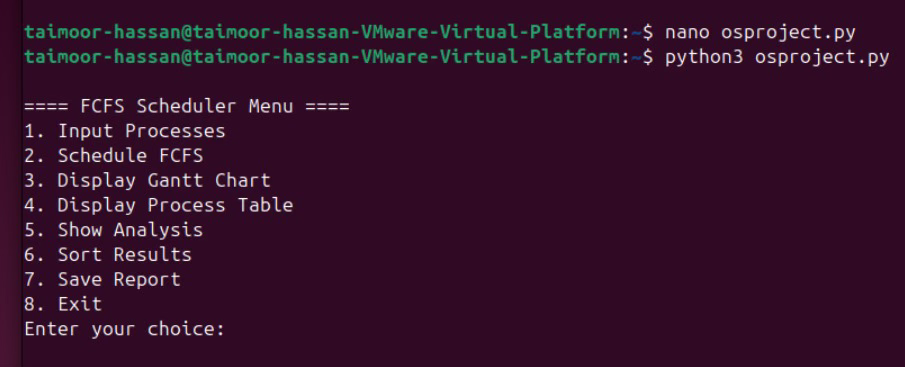
**Average Turnaround and Waiting Time**

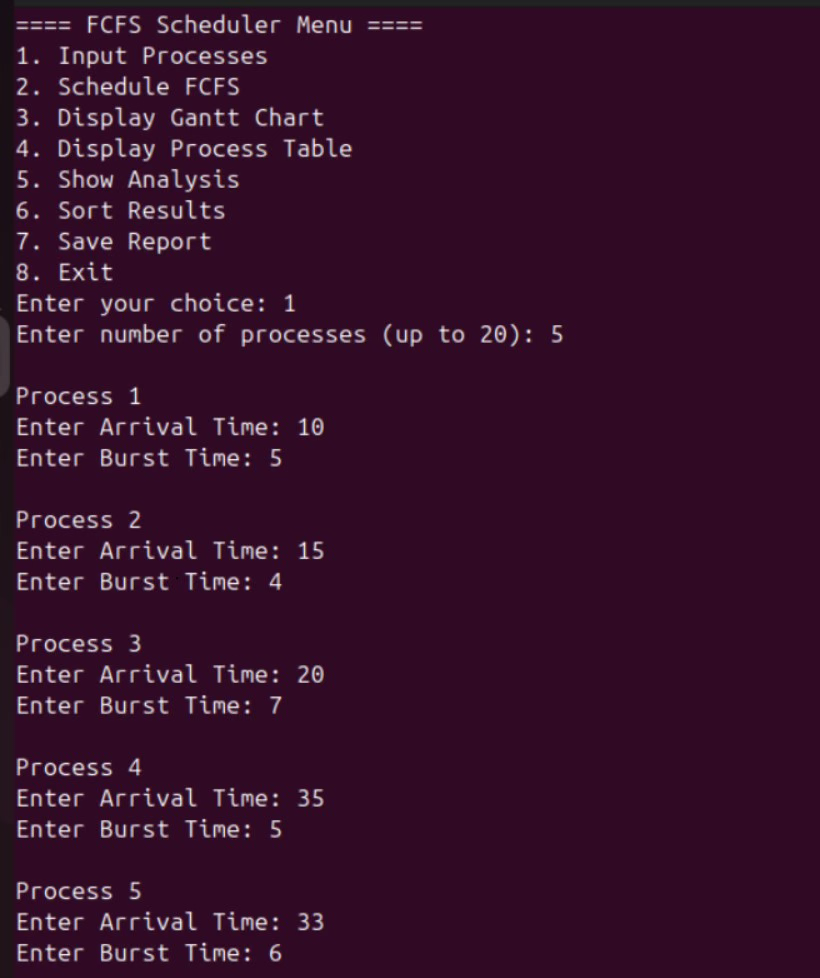
**Gantt Chart** that visually shows when each process starts and ends

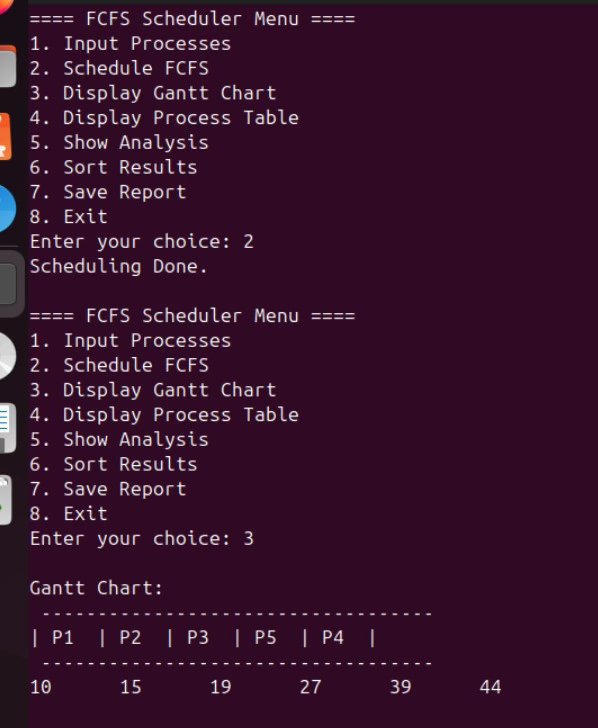
### Algorithm Implementation

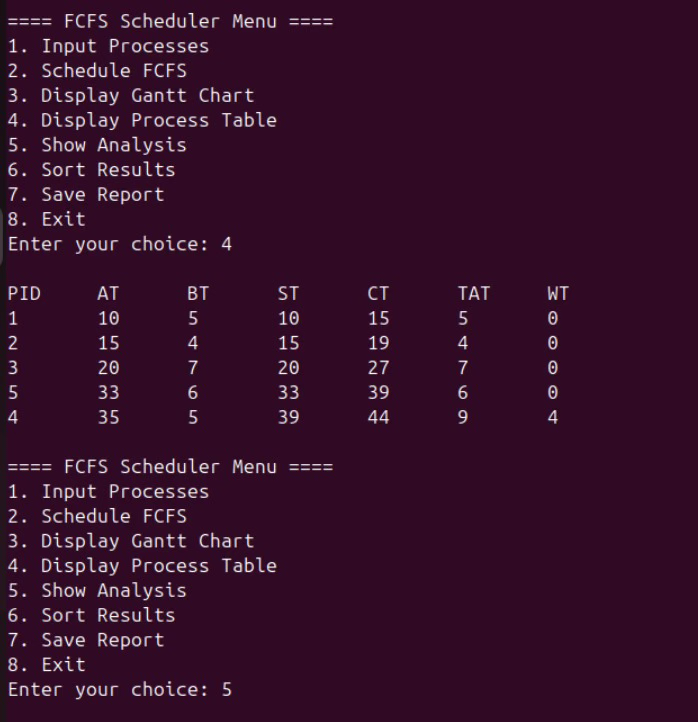
1. Processes are sorted based on **Arrival Time**.
2. The CPU executes each process in the order they arrive.
3. Completion Time is calculated by adding Burst Time to the previous finish time.
4. From Completion Time, the Turnaround Time and Waiting Time are derived.
5. A **Gantt Chart** is drawn using matplotlib to display the process timeline.

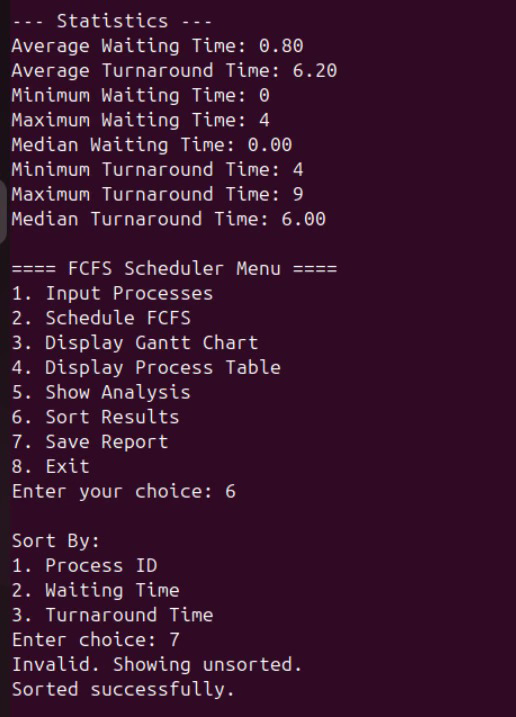
**📸 Output Screenshots**

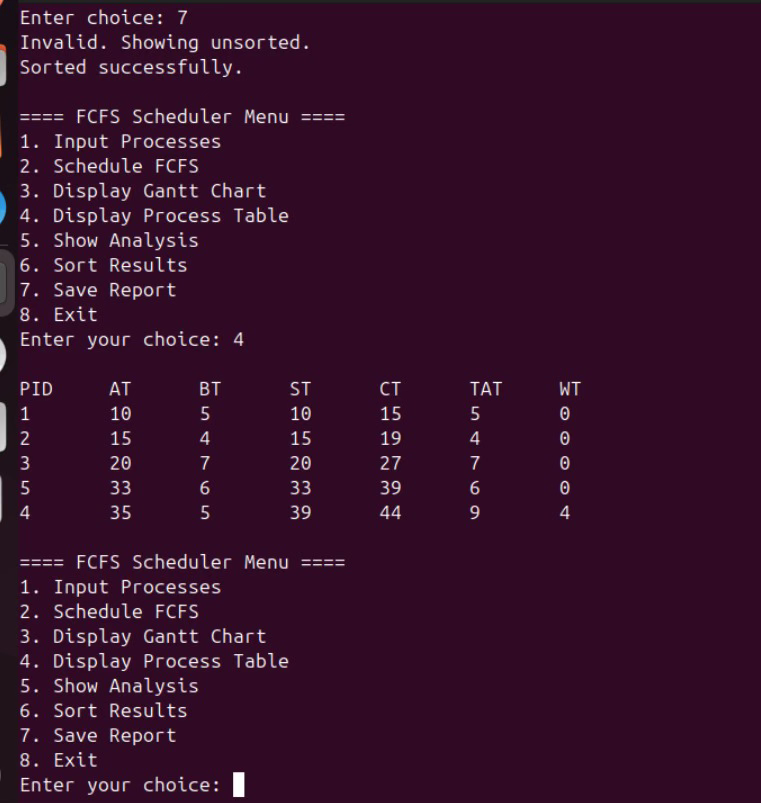












**🧠 Code Structure & Explanation**

1. **Functions Used**

The project is modular and uses well-defined **functions** for each task:

input\_processes():  
Takes user input for number of processes, arrival time (AT), and burst time (BT).

schedule\_fcfs(processes):  
Implements the **core FCFS scheduling logic**, calculates:

Start Time (ST)

Completion Time (CT)

Turnaround Time (TAT)

Waiting Time (WT)

Averages for TAT and WT

display\_gantt\_chart(result):  
Prints a **text-based Gantt Chart** showing process execution order and timing.

display\_table(result):  
Displays a tabular summary of all processes and their calculated timings.

show\_analysis(result, avg\_wt, avg\_tat):  
Prints **min, max, median**, and **average** for turnaround and waiting times.

save\_report(result, avg\_wt, avg\_tat):  
Saves all process data and statistics to a .txt file.

sort\_results(result):  
Allows sorting of output by Process ID, Waiting Time, or Turnaround Time.

menu():  
Main interactive menu for the user to navigate and use all features.

### 2. ****Core Logic of the Scheduling Algorithm****

Processes are **sorted by Arrival Time**.

For each process:

**Start Time** = max(current time, arrival time)

**Completion Time** = Start Time + Burst Time

**Turnaround Time** = Completion Time - Arrival Time

**Waiting Time** = Turnaround Time - Burst Time

These values are stored in a results list for display and analysis.

### 3. ****External Libraries Used****

No external libraries like matplotlib or tabulate are used in this version.

However:

You can use PrettyTable or tabulate for cleaner table output (optional).

matplotlib can be integrated for graphical Gantt chart visualization (if needed in the future).

|  |  |
| --- | --- |
|  |  |

**🛠️ Challenges Faced**

### 1. Incorrect Waiting Time & Turnaround Time Calculation

**Challenge**:  
Initially, the **waiting time (WT)** and **turnaround time (TAT)** values were incorrect, especially when processes had gaps in arrival times (i.e., CPU was idle).

**Solution**:  
We ensured that the **start time (ST)** for each process is set as the **maximum of current time and arrival time**. This allowed accurate handling of CPU idle time and corrected all timing calculations.

### 2. Gantt Chart Alignment

**Challenge**:  
While generating the **text-based Gantt chart**, the formatting and time labels were misaligned, making the output hard to read.

**Solution**:  
We used a consistent structure of ------ and | Pn | blocks, and carefully printed the **start and completion times** with tab spacing to align everything clearly.

### 3. Process Sorting Logic

**Challenge**:  
If processes were not sorted by arrival time before scheduling, the algorithm produced incorrect results and violated FCFS rules.

**Solution**:  
We added a sort() function at the beginning of the scheduling function to always sort processes by **arrival time**before computing results.