Project Title: Shortest Job First (SJF) CPU Scheduling Simulator

Language Used: Python
Author: NEELAM SANGAR
Date: DEC, 31, 2025

1. Introduction

The Shortest Job First (SJF) scheduling algorithm is a CPU scheduling technique in operating systems that selects the process with the smallest CPU burst time from the ready queue. This project develops a console-based SJF CPU Scheduler Simulator in Python. The simulator reads process data from CSV files, calculates key scheduling metrics, generates a Gantt chart, and supports multiple test scenarios. Users can select from multiple CSV files to observe how SJF scheduling optimizes waiting and turnaround times compared to other scheduling algorithms.

2. Objectives

Implement non-preemptive SJF scheduling in Python.

Read process data from CSV files.

Calculate scheduling metrics including Waiting Time (WT) and Turnaround Time (TAT).

Compute average TAT and average WT.

Generate a text-based Gantt chart to visualize process execution.

Support multiple test cases to analyze different scenarios.

3. Key Concepts and Definitions

Process: A program in execution.

CPU Burst Time: The time a process requires the CPU for execution.

Arrival Time: The time at which a process enters the ready queue.

Waiting Time (WT): Total time a process spends waiting before execution. WT = TAT - Burst Time.

Turnaround Time (TAT): Total time from process arrival to completion. TAT = WT + Burst Time.

Gantt Chart: A visual representation of the order and duration of process execution.

CSV File: A Comma-Separated Values file containing process data. Each row represents a process with pid, arrival_time, and burst_time.

4. **Program Structure**

4.1 Dynamic CSV Selection

The program presents a menu of CSV files (sjf_input_case1.csv to sjf_input_case4.csv). Users select a CSV to run; entering 0 exits the program. This allows multiple test runs without restarting. The simulator automatically:

Reads process data into Process objects.

Sorts processes by arrival time.

Selects the shortest burst time from available processes at each CPU time unit.

Updates WT and TAT for each process.

Generates a Gantt chart showing execution order.

CSV Files Used:

sjf_input_case1.csv – Processes with small sequential bursts.

sjf_input_case2.csv – Varied arrival times creating idle CPU gaps.

sjf_input_case3.csv – Processes arriving simultaneously; high contention.

sjf_input_case4.csv – Mixed bursts and arrivals; realistic scenario.

4.2 Python Code Flow

Step 1: CSV Reading and Process Storage

Import csv module.

Maps user choices to CSV file paths.

Reads CSV using csv.DictReader and converts arrival_time and burst_time to integers.

Stores each process as a Process object in a list.

Step 2: Scheduling Logic

Sort processes by arrival time.

For each CPU time unit:

Select processes that have already arrived.

If multiple processes are available, choose the one with the smallest burst time.

If no process has arrived, CPU remains idle.

Step 3: Calculating WT and TAT

For each executed process:

WT = Start Time - Arrival Time

TAT = WT + Burst Time

Step 4: Calculating Average WT and TAT

Sum all WT and TAT values.

Compute averages:

Average_WT = Total_WT / Number_of_Processes

Average_TAT = Total_TAT / Number_of_Processes

Step 5: Displaying Results

Prints a table showing PID, Arrival Time, Burst Time, WT, and TAT.

Prints average WT and TAT.

Step 6: Gantt Chart Generation

Prints execution order of processes including idle times.

Shows timeline with start and end times for each process.

5. Scheduling Results Format

Example output for one CSV input:

PID	Arrival	Burst	WT	TAT
1	0	5	0	5
2	1	3	4	7
3	2	8	7	15
4	3	6	15	21

Average WT: 6.5
Average TAT: 12.0

GANTT CHART:
| 1 (0-5) | 2 (5-8) | 3 (8-16) | 4 (16-22) |

6. Analysis of Different Test Cases

CSV File	Scenario Description	Observations
sjf_input_case1.csv	Processes arrive sequentially with short bursts	Minimal waiting; CPU mostly busy
sjf_input_case2.csv	Some processes arrive later, idle CPU gaps	Waiting time increases for later processes
sjf_input_case3.csv	All processes arrive simultaneously	First process has 0 WT; others accumulate WT based on burst times
sjf_input_case4.csv	Mixed arrivals and bursts	Realistic scenario; demonstrates average TAT and WT variations

Key Insight: SJF favors short processes, minimizing overall waiting time. Longer processes may wait, showing SJF's efficiency in improving average performance but potential starvation risk for long processes.

7. How to Run the Program

Install Python 3.x.

Place CSV files in csv_test_files/SJF_INPUTS/.

Open terminal or VS Code.

Navigate to project folder.

Run: python sjf_scheduler.py

Follow prompts to select a CSV file (1–4).

View results, averages, and Gantt chart.

Repeat with another CSV or exit.

8. Conclusion

The SJF CPU Scheduler Simulator successfully demonstrates non-preemptive SJF scheduling, calculates key metrics, and generates visual Gantt charts. Testing multiple scenarios provides insights into waiting time reduction and CPU efficiency. The simulator serves as a foundation for comparing SJF with other algorithms like FCFS or Round Robin.