FCFS CPU Scheduler Simulator Report

Project Title: First-Come-First-Served (FCFS) CPU Scheduling Simulator

Language Used: Python

Author: PRINCE JOHN MARTINE

Date: JAN,01,2026

1. Introduction

The First-Come-First-Served (FCFS) scheduling algorithm is one of the simplest CPU scheduling algorithms used in operating systems. In this project, a console-based FCFS CPU Scheduler Simulator was developed in Python. The simulator reads process data from CSV files, calculates key scheduling metrics, generates a Gantt chart, and allows for multiple test scenarios.

The simulator allows users to select from multiple test cases to observe the behavior of FCFS scheduling under different process arrival times and burst times.

2. Objectives

1. Implement FCFS scheduling in Python.
2. Read process data from CSV files.
3. Calculate scheduling metrics including Start Time (ST), Completion Time (CT), Turnaround Time (TAT), Waiting Time (WT).
4. Compute average TAT and average WT.
5. Generate a text-based Gantt chart to visualize the execution order.
6. 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 arrives in the ready queue.
- **Start Time (ST):** The time at which a process starts execution. Calculated as the maximum of the previous process's completion time and the current process's arrival time.
- **Completion Time (CT):** The time at which a process finishes execution.
- **Turnaround Time (TAT)**: Total time taken for a process to complete after arrival. TAT = CT - Arrival Time.
- **Waiting Time (WT)**: Total time a process spends waiting in the ready queue. WT = TAT - Burst Time.
- **Gantt Chart:** A visual representation of the scheduling order and duration of processes.
- **CSV File**:A Comma-Separated Values file containing process information. Each CSV row represents a process with its ProcessId, Arrival_Time, and Burst_Time.

4. Program Structure

4.1 Dynamic CSV Selection:

The program presents a menu of CSV files (fcfs_input_1.csv to fcfs_input_4.csv). The user selects which CSV to run. Entering 0 exits the program. This allows multiple test runs in a single execution session.

Automated Process Handling:

The simulator automatically:

- Reads process data into a Python list of dictionaries.
- Sorts processes by arrival time.
- Computes ST, CT, TAT, and WT.
- Generates a Gantt chart.
- Repeatable Execution:

After completing a run, the program returns to the selection menu, allowing the user to run another CSV file without restarting the program.

The simulator reads process information from CSV files. Each CSV contains ProcessId, Arrival_Time, and Burst_Time.

CSV Files Used:

1. fcfs_input_1.csv: Processes arrive sequentially with small bursts.
2. fcfs_input_2.csv: Processes have varied arrival times, some with idle CPU gaps.
3. fcfs_input_3.csv: Processes arrive simultaneously; high contention scenario.
4. fcfs_input_4.csv: Mixed bursts and arrival times to test varied waiting times.

4.2 Python Code Flow

1. User Input: The user selects a CSV file (1–4). The program loads the chosen file.

Step 1: CSV Reading and Process Storage

- Importing the csv module.
- Defines a mapping of choices to CSV file paths.
- Uses a while loop to display the menu and allow multiple runs.
- Reads the chosen CSV from user selection file as a dictionary using csv.DictReader.
- Converts Arrival_Time and Burst_Time from string to integer.
- Stores each process as a dictionary in a list processes.

Step 2: Calculating Start Time and Completion Time

- Sorts processes by Arrival_Time to follow FCFS logic.
- Initializes a variable prev_compTime to keep track of the last process's completion time.
- For each process: ST = max(prev_compTime, Arrival_Time) ,CT = ST + Burst_Time
- Updates prev_compTime after each process finishes.

Step 3: Calculating Turnaround Time (TAT) and Waiting Time (WT)

- For each process:
- TAT = CT – Arrival_Time
- WT = TAT - Burst_Time

Step 4: Calculating Average TAT and WT

- Sums all TATs and Wts.
- Computes averages
- Average_TAT = Total_TAT / number_of_processes
- Average_WT = Total_WT / number_of_processes

Step 5: Displaying Results

- Prints a detailed process table including:
- ProcessId, Arrival_Time, Burst_Time, ST, CT, TAT, WT
- Prints average TAT and WT.

Step 6: Gantt Chart Generation

- Prints process IDs in execution order.
- Prints time markers under each process to indicate execution timeline.

5. Scheduling Results Format

Example output for one CSV input:

ProcessID	P1	P2	P3	P4
Arrival	0	1	2	3
Burst	5	3	8	6
Start	0	5	8	16
Completion	5	8	16	22
TAT	5	7	14	19
WT	0	4	6	13

Average TAT: 11.25

Average WT: 5.75

GANTT CHART:

| P1 | P2 | P3 | P4 |

0 5 8 16 22

6. Analysis of Different Test Cases

CSV File - Scenario Description - Observations

fcfs_input_1.csv: Processes arrive sequentially with short bursts. Minimal waiting times; CPU is almost always busy.

fcfs_input_2.csv: Some processes arrive later, creating idle CPU gaps. Increases waiting time for later processes.

fcfs_input_3.csv: Processes arrive at the same time. Higher contention; first process has 0 WT, others accumulate WT based on FCFS.

fcfs_input_4.csv: Mixed arrivals and bursts. Shows a more realistic scenario; demonstrates average TAT and WT variations.

Key Insight: FCFS favors the first-arriving processes. Processes with later arrival times or longer bursts can significantly increase waiting times, demonstrating the importance of considering alternative scheduling in real-time or interactive systems.

7. How to Run the Program

1. Ensure Python 3.x is installed.
2. Place all CSV files inside csv_test_files/FCFS_INPUTS/.
3. Open terminal or VS Code.
4. Navigate to the project folder.
5. Run: python fcfs_scheduler.py
6. Follow the prompt to select a CSV file (1–4).
7. Results, averages, and Gantt charts will be displayed in the terminal.
8. You can select another CSV file or exit.

8. Conclusion

The FCFS CPU Scheduler Simulator successfully demonstrates process scheduling, computes key metrics, and generates visual Gantt charts. The ability to test multiple scenarios provides insights into CPU utilization, waiting times, and turnaround times. This simulator can serve as a foundation for comparing FCFS with other scheduling algorithms like SJF or Round Robin.