Dynamic CPU Scheduling Simulator

Fahima Lokman Niha
ID : 21070103
Bsc. Computer Science and Engineering
University of science & tecnology, Chittagong

***Abstract*—**The Dynamic CPU Scheduling Simulator is an interactive tool that allows the modeling and execution of the following three basic CPU scheduling algorithms: First-Come-First-Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling. This simulator presents a detailed description of how processes are managed according to any of the different approaches and allows for the evaluation of the performance of the approaches in terms of waiting time, turnaround time and CPU usage among other factors. Furthermore, it explains the reason why the simulator allows users to enter processes specifications so as to help them more clearly see what effect each of the scheduling techniques has on the performance of the CPU as they also view the Gantt charts related to each scheduling algorithm. The purpose of the system is to promote comprehension of the differences between the described two types of scheduling and the tendencies of each one of them in use. Hence, this system is a useful learning and a research measuring device for operating systems.

Keywords—CPU Scheduling, First-Come-First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling, Gantt Chart, Process Management, Waiting Time, Turnaround Time, CPU Utilization, Operating Systems, Performance Analysis, Interactive Simulator, Streamlit, Visualization, matplotlib **.**

I. INTRODUCTION

CPU scheduling is one of the basic principles in OS which is a mechanism that provides the order in which processes will get access to the CPU. This is crucial for the system as it relates to performance, responsiveness and efficiency of the operating system. Due to the fact that in most systems there are several such processes that need to run concurrently especially in the contemporary computer era, it becomes imperative that such processes be managed with a scheduling method that minimizes waiting time, turnaround time among other performance indicators.

This project referred to as the Dynamic CPU Scheduling Simulator is an educational and analytical development that is meant to illustrate and evaluate three basic CPU scheduling techniques - First-Come-First-Serve (FCFS), Shortest Job First (SJF) and Priority Scheduling. Scheduling, in this case, is the assignment of CPU time to processes and each algorithm of the three incorporates a different approach that takes into consideration time of arrival of the process, of the CPU burst time, and priority. In FCFS, the jobs are processed in the order of arrival, SJF scheduling selects processes to be executed next those whose CPU burst time are minimum and Priority Scheduling execution of processes is based on priorities of jobs. Each of the approaches has unique benefits and limitations which make it appropriate for certain cases.

The simulator has been prepared for practical purposes and users are able to provide their own processes and look upon the results of operation of each algorithm. The users can also see the performance of different processes in the different turns of the CPU and how the performance is controllable by the scheduling of the processes. Additional key metrics such as waiting time and turnaround time are derived and presented to the users to visualize the impact of the change of algorithms. The Dynamic CPU Scheduling Simulator serves as both a learning tool and a performance analysis platform, offering insights into the behavior of these scheduling strategies under various workloads and process configurations. By presenting real-time data in the form of Gantt charts and comparative reports, the simulator facilitates a deeper understanding of CPU scheduling's critical role in operating systems.

II. BACKGROUND

A. Project : Create a CPU Scheduling Simulator to showcase and compare the performance of three key scheduling algorithms: Shortest Job First (SJF), First-Come-First-Serve (FCFS), and Priority Scheduling. This simulator will help analyze and visualize the impact of each algorithm on CPU performance metrics, such as average waiting time, turnaround time, and CPU utilization under different conditions. *Project Goals:*

1. Develop implementations for the Shortest Job First (SJF), First-Come-First-Serve (FCFS), and Priority scheduling algorithms.
2. Enable users to enter process details, including burst time, arrival time, and priority, for each scheduling scenario.
3. Provide visualizations for each algorithm, illustrating the sequence of process execution along with individual waiting and turnaround times.
4. Produce a comparative analysis report to evaluate scheduling metrics across algorithms, highlighting differences in performance.

B. Simulator: The CPU Scheduling Simulator is designed to visually and interactively demonstrate the functionality of different CPU scheduling algorithms, namely First-Come-First-Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling. Through a user-friendly interface, users can input process attributes—such as arrival time, burst time, and priority level—to simulate various scheduling scenarios.

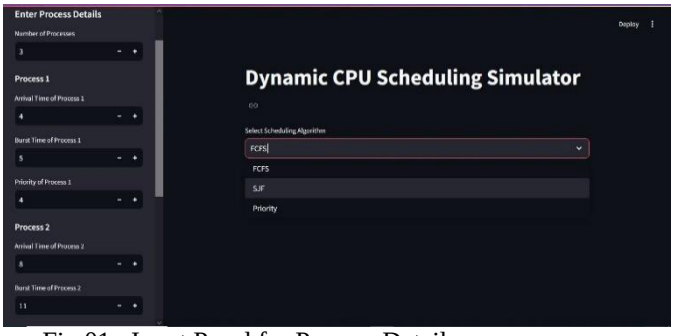

Fig 01 : Input Panel for Process Details

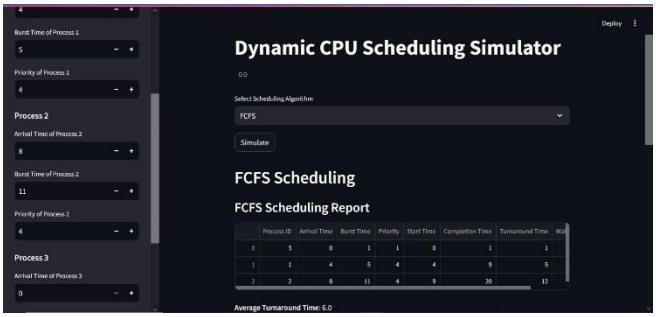

Fig 02 : FCFS Scheduling Report

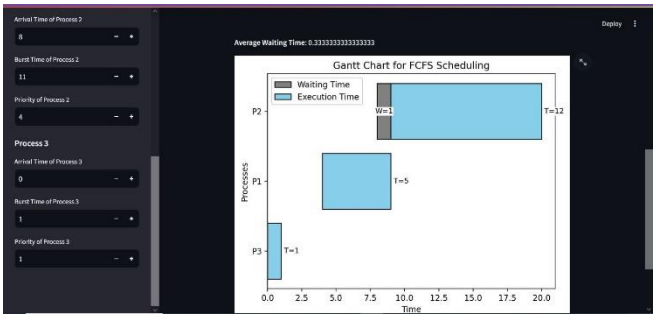

Fig 03: FCFS Scheduling Gantt Chart

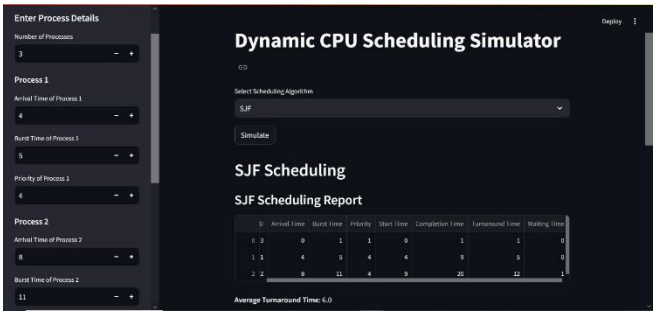

Fig 04 : SJF Scheduling Report.

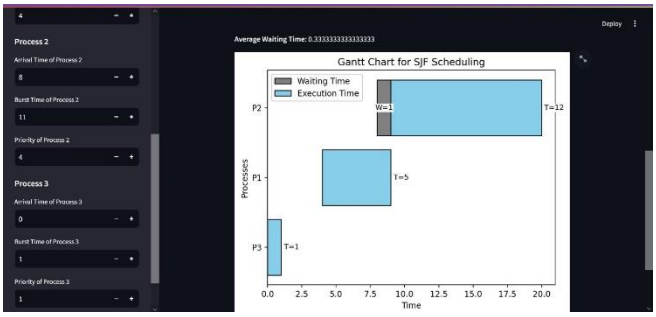

Fig 05: SJF Scheduling Gantt Chart.

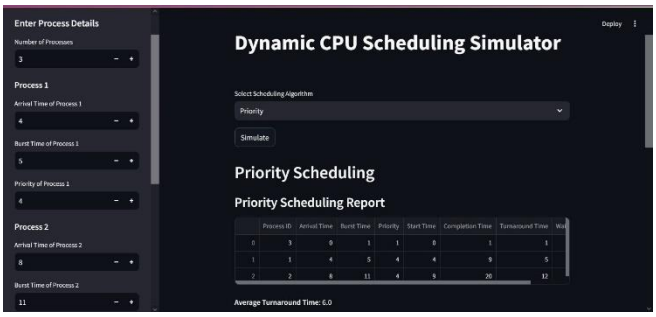

Fig 06: Priority Scheduling Report .

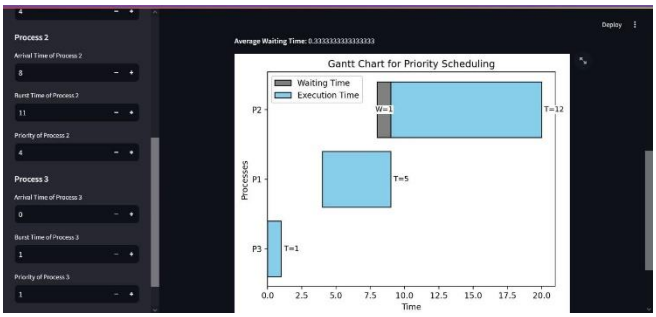

Fig 07 : : Priority Scheduling Gantt Chart

III. PROJECT EVALUATION

C. Setup Environment : To run the CPU Scheduling Simulator, follow these steps:

1. **System Requirements**: Ensure that Python 3.7 or higher is installed. Required libraries include matplotlib for charts and numpy for calculations.
2. **Library Installation**: Install necessary libraries by running: pip install matplotlib numpy
3. **Running the Simulator**: Go to the folder containing the simulator code and run the main file to start. For a web-based version, start the server (e.g., with Flask) and open the link provided in a browser.
4. **Configuration**: Adjust process parameters and choose the scheduling algorithm as needed within the simulator.

This setup enables users to quickly input data, view Gantt charts, and analyze scheduling performance metrics.

D. Code in Simulator :

```
# Define the Process Class
Class Process:
  Initialize with:
      pid          # Unique identifier for the process
      arrival_time   # Time when the process arrives in the system
      burst_time     # Time required to complete the process
      priority       # Priority level (lower value indicates higher priority)
      completion_time = 0     # Time when the process completes execution
      turnaround_time = 0     # Total time from arrival to completion
```

```
    waiting_time = 0       # Total time the process spends waiting
    start_time = 0         # Time when the process starts execution
# First-Come-First-Serve (FCFS) Scheduling Function
Function FCFS(processes):
  Sort processes by arrival_time in ascending order
  Set current_time = 0
For each process in the sorted list:
    Set process.start_time to the maximum of current_time and process.arrival_time
    Calculate process.completion_time as process.start_time + process.burst_time
    Calculate process.waiting_time as process.start_time - process.arrival_time
    Calculate process.turnaround_time as process.completion_time - process.arrival_time
    Update current_time to process.completion_time
Return processes with updated scheduling metrics
# Shortest Job First (SJF) Scheduling Function
Function SJF(processes):
  Sort processes by arrival_time, then by burst_time for same arrival time
  Set current_time = 0
  Set completed = 0
  Set total_processes = length of processes
While completed < total_processes:
    Find available_processes with arrival_time <= current_time and not completed
    If available_processes is not empty:
      Select process with the minimum burst_time
      Set process.start_time to max(current_time, process.arrival_time)
      Calculate process.completion_time as process.start_time + process.burst_time
      Calculate process.waiting_time as process.start_time - process.arrival_time
      Calculate process.turnaround_time as process.completion_time - process.arrival_time
      Update current_time to process.completion_time
      Increment completed by 1
    Else:
      Increment current_time by 1
Return processes with updated scheduling metrics
# Priority Scheduling Function
Function PriorityScheduling(processes):
  Sort processes by arrival_time, then by priority (lower value indicates higher priority)
  Set current_time = 0
For each process in sorted list:
    Set process.start_time to max(current_time, process.arrival_time)
    Calculate process.completion_time as process.start_time + process.burst_time
    Calculate process.waiting_time as process.start_time - process.arrival_time
    Calculate process.turnaround_time as process.completion_time - process.arrival_time
    Update current_time to process.completion_time
 Return processes with updated scheduling metrics
# Calculate Metrics Function
Function CalculateMetrics(processes):
  For each process in processes:
    Calculate process.turnaround_time as process.completion_time - process.arrival_time
    Calculate process.waiting_time as process.turnaround_time - process.burst_time
Return processes with updated metrics
# Gantt Chart Visualization Function
Function VisualizeGanttChart(processes, algorithm_name):
  Initialize Gantt chart plot
For each process in processes:
    Plot waiting time from process.arrival_time to process.start_time (in grey)
    Plot execution time from process.start_time to process.completion_time (in blue)
    Annotate waiting time (W) and turnaround time (T)
Set chart title to "Gantt Chart for {algorithm_name} Scheduling"
  Display Gantt chart
# Display Metrics Report Function
Function PrintReport(processes, algorithm_name):
  Prepare table with columns:
    Process ID, Arrival Time, Burst Time, Priority, Start Time, Completion Time, Turnaround Time, Waiting Time
  Display the table
  Calculate and display average turnaround time and waiting time
# Streamlit Application Interface
Initialize Streamlit App with Title: "Dynamic CPU Scheduling Simulator"
Sidebar:
  Accept input for number of processes (num_processes)
  For each process:
    Accept input for arrival_time, burst_time, priority
    Append process to processes list
Main Panel:
  Display dropdown for scheduling algorithm (FCFS, SJF, Priority)
  If "Simulate" button clicked:
    Run selected scheduling function (FCFS, SJF, or Priority)
    Display algorithm name

    Call CalculateMetrics(processes)
    Call PrintReport(processes, algorithm_name)
    Call VisualizeGanttChart(processes, algorithm_name)
End Streamlit App
```

Key features of this code sre :

- Input Interface: Accepts process details (ID, Arrival Time, Burst Time, Priority) and supports multiple processes. Key functions include scheduling (fcfs, sjf, priority_scheduling), metrics calculation, report display, and Gantt chart visualization.
- Scheduler Module: Implements FCFS (by arrival), SJF (shortest burst), and Priority Scheduling (highest priority first).
- Simulation and Visualization: Displays scheduling order with Gantt charts, and shows each process's waiting, turnaround, and response times, enabling visual comparison across algorithms.

IV. Critical Evaluation

The simulator effectively demonstrates how FCFS, SJF, and Priority Scheduling impact CPU performance metrics like waiting and turnaround times. FCFS is simple but can cause long wait times for shorter jobs. SJF minimizes average waiting time but risks starving longer processes**.** Priority scheduling ensures critical tasks are prioritized but can starve lower-priority jobs. Overall, each algorithm has unique strengths and weaknesses, making the simulator valuable for understanding these trade-offs.

The Dynamic CPU Scheduling Simulator provides an interactive, visual approach to understanding CPU scheduling algorithms. By comparing FCFS, SJF, and Priority Scheduling, users can observe the impact on performance metrics, gaining insights into the practical applications and limitations of each algorithm. This simulator serves as an educational tool for better grasping CPU scheduling concepts.

V. Acknowledgment

We would like to thank our instructors and peers for their guidance and support in developing this simulator, which has enhanced our understanding of CPU scheduling principles and programming.

VI. References

1. Rahman, M., & Khan, A. (2024). Comparative Analysis of CPU Scheduling Algorithms in Real-Time Systems. *Journal of Computer Science*, 20(6), 972-985. Retrieved from https://thescipub.com/abstract/10.3844/jcssp.2024.972.985

2. Singh, R., & Verma, S. (2021). Analysis of Priority Scheduling Algorithm on the Basis of FCFS & SJF for Similar Priority Jobs. *ResearchGate*. Retrieved from https://www.researchgate.net/publication/354462497_Analysis_of_Priority_Scheduling_Algorithm_on_the_Basis_of_FCFS_SJF_for_Similar_Priority_Jobs

3. Mazieres, D. (2009). *Lecture Notes on CPU Scheduling*. Stanford University. Retrieved from https://www.scs.stanford.edu/09wi-cs140/notes/l5.pdf

4. Shiksha.com. (n.d.). CPU Scheduling Algorithms in Operating Systems. Retrieved from https://www.shiksha.com/online-courses/articles/cpu-scheduling-algorithm-operating-system/