CPU SCHEDULING SIMULATOR

A MINI-PROJECT REPORT

Submitted by

VAISHNAVI E	**231901059**
YARRA JAISURYA	**231901063**

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

(CYBER SECURITY)

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY

CHENNAI – 600 025

MAY 2025

BONAFIDE CERTIFICATE

Certified that this project **“CPU SCHEDULING SIMULATOR”** is the bonafide work of **“VAISHNAVI E & YARRA JAISURYA”** who carried out the project work under my supervision.

SIGNATURE
Mrs.V.JANANEE
ASSISTANT PROFESSOR(SG)

Department of Computer Science
and Engineering,
Rajalakshmi Engineering College,
Chennai

This mini project report is submitted for the viva voce examination to be held on ___________

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our sincere thanks to our Head of the Department **Mr.S.MEGANATHAN** and the Chairperson **Dr.M.THANGAM MEGANATHAN** for Their timely support and encouragement.We are greatly indebted to our respected and honourable principal **Dr. S.N.MURUGESAN,** For his able support and guidance.We express our sincere thanks to our Head of the Department **Mr. Benedict JN,** for encouragement and being ever supporting force during our project work.We also extend our sincere and hearty thanks to our internal guide **Mrs.V.JANANEE,** for her valuable guidance and motivation during the completion of this project.Our sincere thanks to our family members, friends and other staff members of computer science engineering.

1. VAISHNAVI E 231901059

2. YARRA JAISURYA 231901063

ABSTRACT

CPU scheduling is a key function in operating systems, directly impacting performance and responsiveness. This mini-project introduces an interactive CPU Scheduling Simulator that visualizes how algorithms like FCFS, SJF, and Priority Scheduling manage processes. Users can input custom data and view real-time Gantt charts and performance metrics like turnaround time. The simulator's intuitive interface and animated visuals make complex concepts easy to grasp. Designed as a learning aid, it bridges the gap between theory and practical understanding of CPU scheduling techniques.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

This project focuses on simulating CPU scheduling in operating systems, aiming to enhance understanding of how different scheduling algorithms impact system performance. The simulator allows users to visualize and compare algorithms like FCFS, SJF, and Priority Scheduling. By providing a graphical and interactive environment, it helps learners grasp the logic behind process execution and CPU resource management.

1.2 SCOPE OF THE WORK

The CPU Scheduling Simulator serves as an educational tool to demonstrate the behavior of scheduling algorithms under various conditions. It enables users to input custom process data, view animated Gantt charts, and analyze turnaround and waiting times. The project can be used by students, educators, and developers to better understand scheduling mechanics and performance impacts.

1.3 PROBLEM STATEMENT

Understanding CPU scheduling is critical but often difficult due to its abstract nature and lack of visual representation. Many learners struggle to connect theoretical concepts with practical execution. This project addresses that gap by offering an intuitive, visual way to simulate and evaluate different CPU scheduling algorithms, making complex topics easier to comprehend.

1.4 AIM AND OBJECTIVES OF THE PROJECT

The aim of this project is to develop a user-friendly simulator that effectively illustrates the functioning of different CPU scheduling algorithms. The project seeks to implement algorithms such as First Come First Serve (FCFS), Shortest Job First (SJF), and Priority Scheduling within an interactive interface. It allows users to enter process details, observe how the CPU schedules them, and evaluate the performance based on metrics like turnaround time and waiting time. The ultimate goal is to simplify the learning process and bridge the gap between theoretical knowledge and practical application.

CHAPTER 2

SCOPE OF THE PROJECT

The CPU Scheduling Simulator offers an interactive environment to visualize and analyze core CPU scheduling algorithms including FCFS, SJF, Priority, and Round Robin. Designed with an educational focus, the tool helps users understand how different strategies affect process execution, turnaround time, and system responsiveness.

Scheduling Strategy Implementation

- **Simulates essential scheduling algorithms:**
  - First-Come-First-Serve (FCFS): Simple, arrival-time-based execution.
  - Shortest Job First (SJF): Optimizes turnaround by executing the shortest processes first.
  - Priority Scheduling: Executes processes based on defined priority levels.
  - Round Robin (RR): Time-sliced scheduling ideal for time-sharing systems.

- **Each algorithm's impact on performance is visualized through:**
    - Gantt charts showing real-time process execution.
    - Comparative performance graphs for turnaround and waiting times.

Interactive GUI and Visualization

- **Tkinter-based GUI allows:**
    - Easy input of process details (arrival, burst, priority).
    - Selection and execution of scheduling algorithms.
    - Real-time Gantt chart visualization for process scheduling.
- **Matplotlib integration provides:**
    - Live performance comparison across algorithms.

- Insightful bar charts of average turnaround and waiting times.

User Customization and Flexibility

- **Users can configure:**
    - Number of processes.
    - Arrival, burst, and priority values.
    - Scheduling algorithm selection dynamically.

- Supports varied simulation setups for in-depth comparative analysis.

Educational and Research Relevance

- Designed for:
    - Students learning operating system concepts like scheduling, fairness, and responsiveness.
    - Educators as a visual aid in classroom demonstrations.
    - Researchers for experimentation with process scheduling strategies.
- Provides foundational understanding of:
    - Context switching and CPU resource allocation.
    - Fairness and starvation issues.
    - Impact of scheduling on system performance.

Performance Analysis and Reporting

- **Live metrics allow users to:**
    - Compare scheduling policies.
    - Analyze efficiency and responsiveness under different scenarios.

- **Offers concrete data for:**
    - Decision-making in algorithm selection.
    - Deeper exploration of scheduling theory

CHAPTER 3

ARCHITECTURE

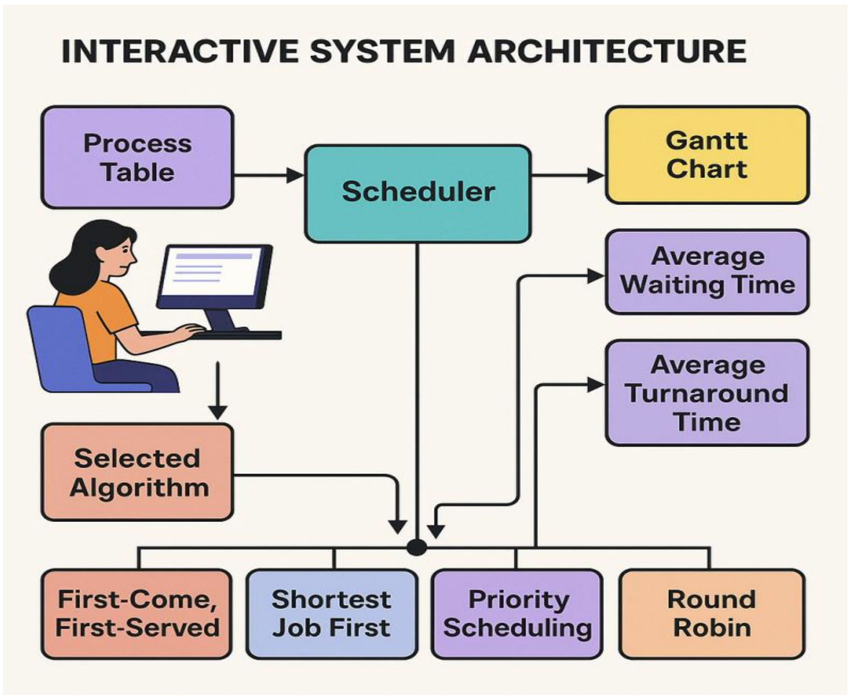


Figure 1. CPU Scheduling Architecture

- **Process Table:** Stores all the processes with their details like arrival and burst time.

- **Scheduler:** Manages and selects processes based on the chosen scheduling algorithm.

- **Gantt Chart:** Visual representation of the process execution timeline.

- **Average Waiting Time:** Calculates the mean time processes wait in the ready queue.

- **Average Turnaround Time:** Measures the average time from process submission to completion.
- **Selected Algorithm:** The user-chosen scheduling method for process execution.
- **First-Come, First-Served (FCFS**): Processes are executed in the order they arrive.
- **Shortest Job First (SJF):** Executes the process with the smallest burst time next.
- **Priority Scheduling:** Executes processes based on priority levels.
- **Round Robin:** Assigns each process a fixed time slot in cyclic order.

CHAPTER 4

FLOWCHART

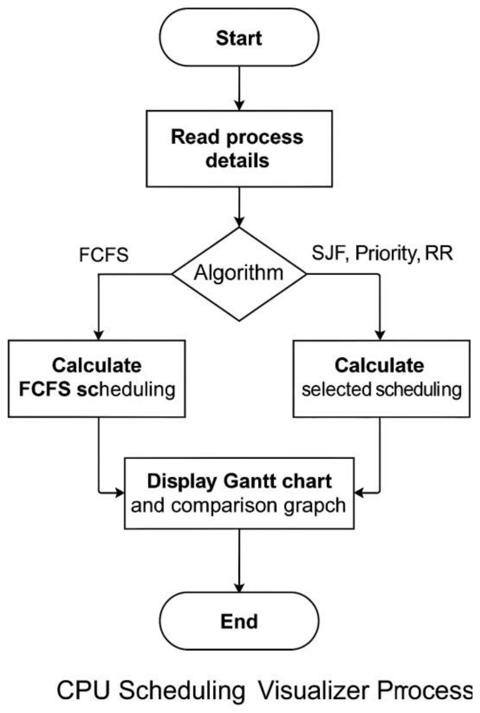

Start
Read process details
FCFS
Algorithm
SJF, Priority, RR
Calculate FCFS scheduling
Calculate selected scheduling
Display Gantt chart and comparison grapch
End
CPU Scheduling Visualizer Prrocess

Figure 2: Flowchart of CPU Scheduling

1. Start

- This is the entry point of the application.
- It signifies the beginning of the scheduling process when the program is launched or the user clicks "Run Scheduler".

2. Read process details

- The user inputs details like:
  - **Arrival Time** (when a process enters the ready queue),
  - **Burst Time** (how long it needs the CPU),
  - **Priority** (if applicable).

3. Algorithm decision block

- **A decision is made based on the selected algorithm:**
  - **FCFS** (First-Come, First-Served)
  - **SJF** (Shortest Job First)
  - **Priority**
  - **RR** (Round Robin)

4A. If FCFS is selected → Calculate FCFS scheduling

- This block handles the **FCFS scheduling logic**.
- Processes are sorted by arrival time and executed in that order.
- Waiting time, turnaround time, start, and completion times are calcul

4B. If SJF, Priority, or RR → Calculate selected scheduling

- This block is for:
  - **SJF**: Selects the shortest burst time among ready processes.
  - **Priority**: Selects the process with the highest priority (lowest number).
  - **RR**: Executes each process for a fixed time quantum in a circular queue.

5. Display Gantt chart and comparison graph

- After scheduling:
  - A **Gantt chart** is generated to visualize the execution timeline.
  - A **bar graph** compares average waiting and turnaround times across all algorithms (FCFS, SJF, Priority, RR).

6. End

- This indicates the **completion** of the scheduling visualization.
- The user can then rerun the scheduler or modify inputs

CHAPTER 5
CODE IMPLEMENTATION

5.1. PROGRAM

5.1.1. CPU SCHEDULING SIMULATION

```
def __init__(self, root):
    # Initializes the GUI and setup

def create_widgets(self):
    # Creates the GUI widgets for the scheduler

def calculate_fcfs(self, processes):
    # First-Come-First-Serve algorithm logic

def calculate_sjf(self, processes):
    # Shortest Job First algorithm logic

def calculate_priority(self, processes):
    # Priority Scheduling algorithm logic

def calculate_rr(self, processes, quantum=2):
    # Round Robin Scheduling algorithm logic

def run_scheduling(self):
    # Fetches input, runs selected scheduling algorithm and visualizes results

def plot_static_gantt_chart(self, processes, title, is_rr=False):
    # Plots the Gantt chart for the selected algorithm

def show_comparison_graph(self, all_avgs):
    # Shows a bar graph comparing average waiting and turnaround t
```

CHAPTER 6

RESULTS

6.1 SCREENSHOTS

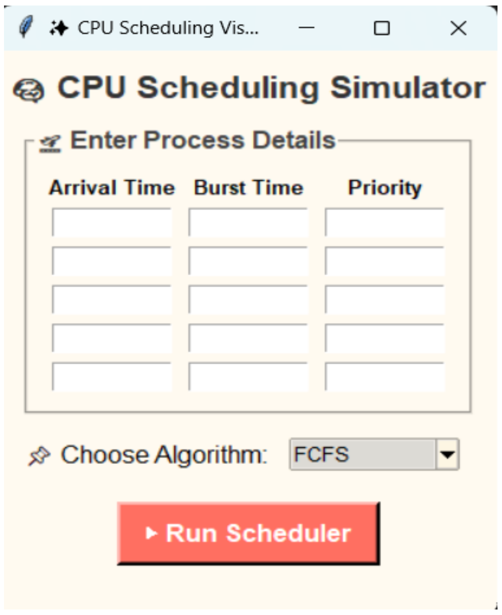
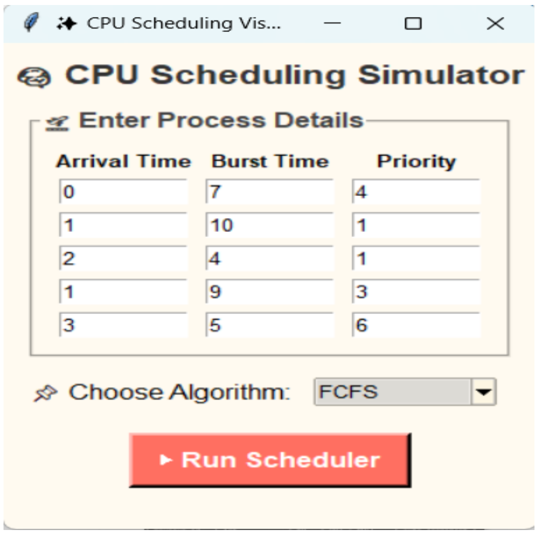


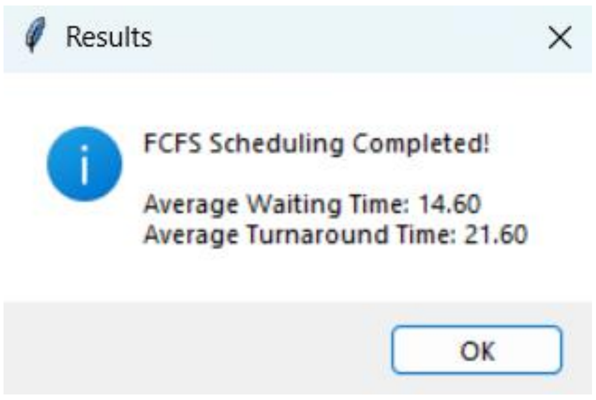
Fig 3. Input Dialog box



Arrival Time	Burst Time	Priority
0	7	4
1	10	1
2	4	1
1	9	3
3	5	6

Choose Algorithm: FCFS

Fig 4. Giving user input



Results

FCFS Scheduling Completed!

Average Waiting Time: 14.60
Average Turnaround Time: 21.60

Fig 5. FCFS Efficiency Log

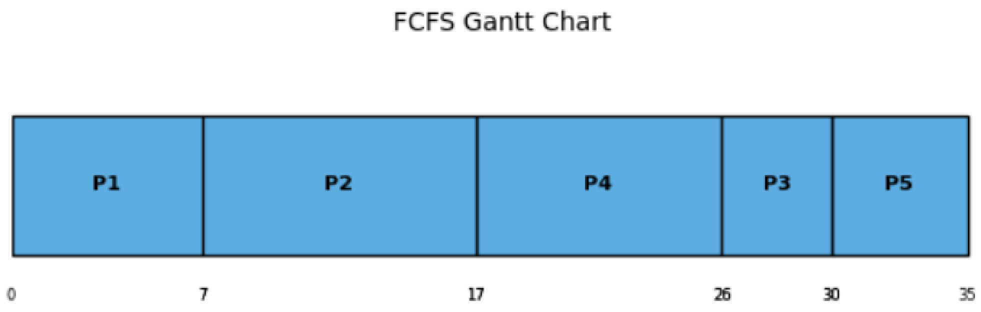
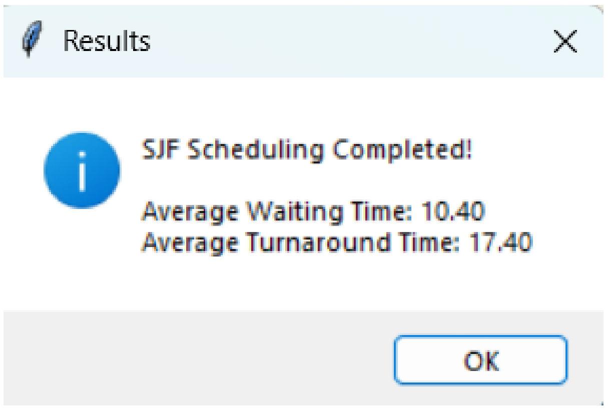


Fig 6. FCFS Gant Chart



Results

SJF Scheduling Completed!

Average Waiting Time: 10.40
Average Turnaround Time: 17.40

Fig 7. SJF Efficiency Log

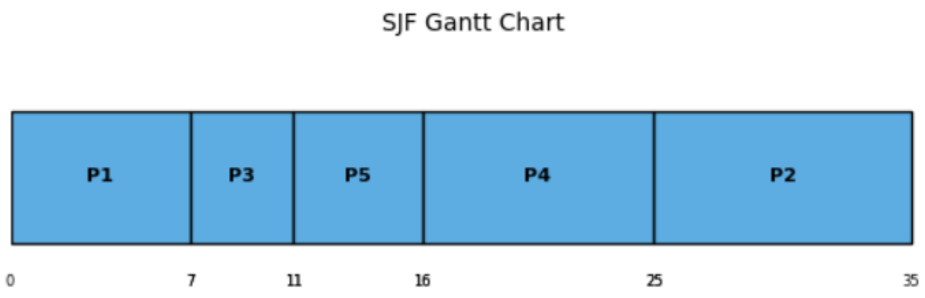
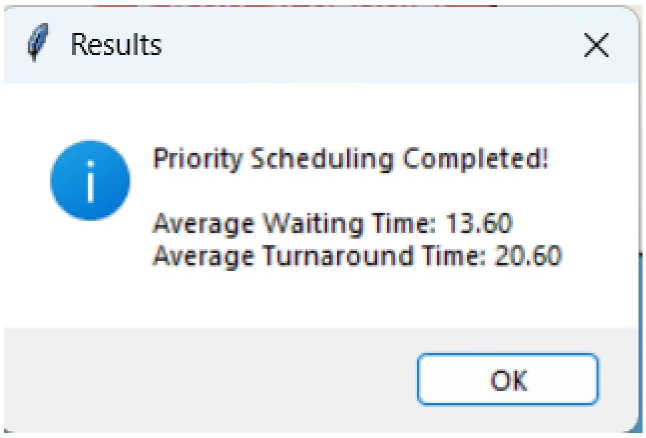


Fig 8. SJF Gant Chart


Results

Priority Scheduling Completed!

Average Waiting Time: 13.60
Average Turnaround Time: 20.60

Fig 9. Priority Scheduling Efficiency Log

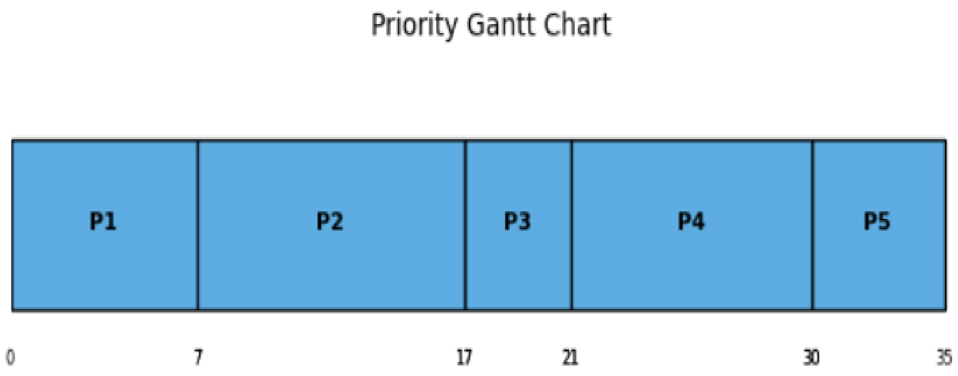
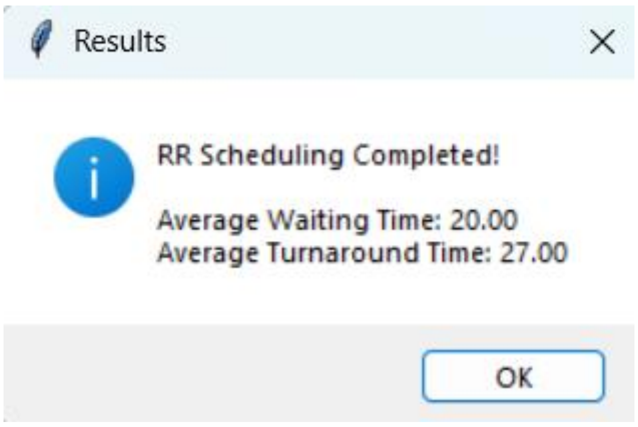


Fig 10. Priority Scheduling Gant Chart



RR Scheduling Completed!

Average Waiting Time: 20.00
Average Turnaround Time: 27.00

Fig 11. RR Efficiency Log

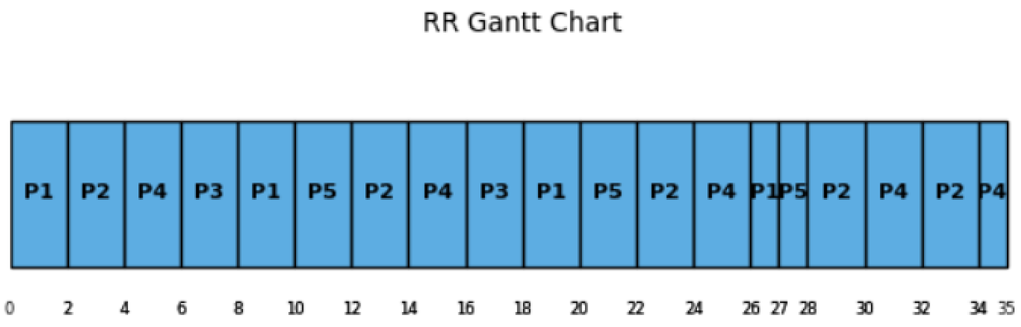


Fig 12. RR Gant Chart

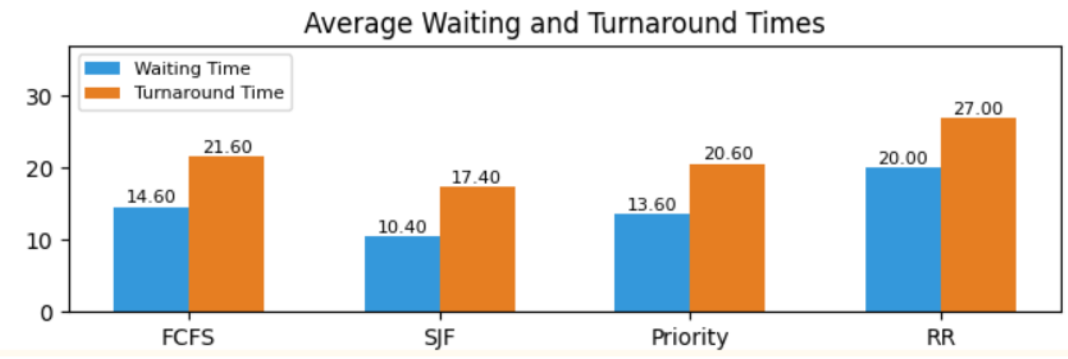


Fig 13. Efficiency Graph

CHAPTER 7

CONCLUSION

The CPU Scheduling Visualizer project provides an insightful and interactive platform for exploring fundamental operating system scheduling algorithms including

FCFS, SJF, Priority Scheduling, and Round Robin. By combining theoretical scheduling principles with a visual and real-time graphical interface built using Python's Tkinter and Matplotlib, this project bridges the gap between textbook learning and practical understanding.

The simulation demonstrates the internal mechanics of CPU scheduling, showcasing how each algorithm affects process waiting time, turnaround time, and execution order. Through vivid Gantt chart animations and comparative performance graphs, learners gain a deeper appreciation of the efficiency and fairness trade-offs in process management.

Incorporating interactive GUI components, customizable process inputs, and clear visual feedback, the project not only enhances the educational experience but also acts as a powerful toolkit for instructors, students, and enthusiasts to experiment, visualize, and analyze CPU scheduling strategies.

Ultimately, this project fosters active learning and strengthens core concepts of CPU scheduling, system performance analysis, and real-time visualization in operating systems.

CHAPTER 8

FUTURE WORK

- **AI-Based Scheduling Enhancements**
  Implement machine learning models to dynamically choose the most efficient scheduling algorithm based on historical process data and system load.

- **Real-Time System Integration**
  Connect the simulator with live system data (e.g., OS process tables or cloud VM loads) for real-time CPU scheduling visualization.

- **Dynamic Scheduling Algorithms**
  Incorporate more advanced and adaptive algorithms like Multilevel Feedback Queue, EDF (Earliest Deadline First), or AG (Adaptive General) scheduling.

- **Web-Based Deployment**
  Develop a web version of the simulator using frameworks like Flask or Django to enable cross-platform, browser-based interaction.

- **Mobile Application Support**
  Extend the project into a mobile-friendly version for on-the-go access and learning.

- **Cloud Hosting and Collaboration**
  Deploy the simulator on cloud platforms with shared access for group learning, classroom demonstrations, and remote experimentation.

- **Simulation of Multiprocessor Scheduling**
  Enhance the simulator to support multi-core CPU environments and simulate parallel scheduling scenarios.

- **Gamification for Learning**
  Add challenge modes or quizzes based on scheduling theory to make the simulator more engaging for students.

- **User Role Management**
  Integrate role-based access for administrators, instructors, and learners to manage input permissions and access simulation logs.

- **Data Export and Report Generation**
  Allow exporting of simulation results (e.g., Gantt charts, performance stats) to PDF or Excel for reporting and further analysis.

- **Accessibility Features**
  Include support for screen readers, high-contrast themes, and keyboard-only navigation to make the tool inclusive.

REFERENCES

1. William Stallings, (2018) “Operating Systems – Internals and Design Principles”, 9thEdition, Pearson.

2. Pavel Y., Alex I., Mark E., David A., (2017)“Windows Internal Part I - System Architecture, Processes, Memory Management and More”, 7th Edition, Microsoft Press.

3. Andrew S. Tanenbaum and Herbert Bos, (2016) “Modern Operating Systems”, 4th Edition, Pearson.