A SYNOPSIS ON

CPU SCHEDULER

Submitted in partial fulfilment of the requirement for the award of the degree of

BACHELOR OF TECHNOLOGY

In

Computer Science & Engineering

Submitted by:

Vansh Pal	**2261591**
Abhishek Bora	**2261065**
Vinayak Joshi	**2261610**
Kuldeepak Singh Khati	**2261329**

Under the Guidance of

Mr. Prince Kumar

Assistant Professor

Project Team ID: 85

Department of Computer Science & Engineering

Graphic Era Hill University, Bhimtal, Uttarakhand

March-2025

Graphic Era Hill University
BHIMTAL CAMPUS

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the Synopsis entitled **"CPU SCHEDULER"** in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science & Engineering of the Graphic Era Hill University, Bhimtal campus and shall be carried out by the undersigned under the supervision of **Mr. Prince Kumar, Assistant Professor**, Department of Computer Science & Engineering, Graphic Era Hill University, Bhimtal.

Vansh Pal	2261591
Abhishek Bora	2261065
Vinayak Joshi	2261610
Kuldeepak Singh Khati	2261329

The above mentioned students shall be working under the supervision of the undersigned on the **"CPU SCHEDULER"**

Supervisor **Head of the Department**

Internal Evaluation (By DPRC Committee)

Status of the Synopsis: Accepted / Rejected

Any Comments:

Name of the Committee Members: **Signature with Date**

1.

2.

Table of Contents

Chapter 1

Introduction and Problem Statement

In the following sections, a brief introduction and the problem statement for this work are included.

1.1 Introduction

CPU scheduling is a vital component of an operating system's functionality, tasked with determining the sequence in which processes in the ready queue are granted access to the central processing unit (CPU). In a multitasking environment, multiple processes may be ready to execute, but only one can occupy the CPU at any given time. Efficient CPU scheduling not only enhances system performance but also ensures fair resource distribution and optimal CPU utilization. With the rapid advancement of modern computing and the increasing demand for responsive, scalable, and efficient systems, the importance of mastering CPU scheduling concepts has become more crucial than ever.

There are several types of CPU scheduling algorithms, each with its own strengths and trade-offs. These include First-Come, First-Served (FCFS), which is simple and easy to implement but can lead to long waiting times; Shortest Job First (SJF), which minimizes average turnaround time but is susceptible to starvation; Round Robin (RR), which is more equitable and suitable for time-sharing systems but depends heavily on time quantum; and Priority Scheduling, which executes processes based on their priority level but may cause low-priority processes to be indefinitely delayed. The selection of the appropriate scheduling algorithm can significantly affect overall system efficiency and user satisfaction.

In traditional learning environments, CPU scheduling is taught through static diagrams and theoretical examples, often presented through textbooks and classroom lectures. While these methods are essential for conveying foundational knowledge, they are insufficient for conveying the dynamic nature of scheduling in a real operating system. Visual learning has proven to be an effective pedagogical approach, especially in technical subjects. Students gain a better understanding when they can observe the behavior of scheduling algorithms through visual simulations.

This project addresses that educational need by introducing an interactive visualization tool for CPU scheduling algorithms. The project will provide an intuitive and user-friendly interface where learners can input custom process details—such as arrival time, burst time, and priority—and see how different scheduling algorithms execute those processes. Real-time Gantt charts will be used to visually represent process execution order, duration, and overlap, giving users a clear and comprehensive view of how the scheduler works.

In addition to visualization, the tool will provide tabulated results including average waiting time, turnaround time, and completion time. This allows users to evaluate and compare the effectiveness of different algorithms under varying conditions. By offering both analytical data and graphical representation, the tool supports multiple learning styles and encourages experimentation.
Ultimately, the aim of this project is to bridge the gap between theoretical understanding and practical application of CPU scheduling. It will serve as a learning aid for students, a teaching tool for educators, and a demonstration platform for system developers interested in scheduling logic. Through this interactive simulator, users will not only understand how scheduling algorithms function but also gain insight into their impact on system performance, responsiveness, and fairness.

1.2. Problem Statement

Despite being a fundamental concept in operating systems, CPU scheduling remains one of the more challenging topics for students to grasp. The abstract nature of scheduling algorithms and the complexity involved in understanding their practical implications often hinder students' ability to fully appreciate the nuances of process management. Traditional educational resources such as textbooks, static flowcharts, and theoretical exercises provide the foundational concepts but lack the dynamic interaction and real-world simulation that are critical to understanding how scheduling actually works.

The primary challenge arises from the inability to visualize the behavior of scheduling algorithms over time. Concepts like preemption, context switching, waiting time accumulation, and the fairness of scheduling strategies are difficult to internalize when presented solely through text or simple diagrams. Furthermore, most existing academic tools do not offer the flexibility to customize input data or dynamically compare multiple scheduling strategies in real-time. As a result, students often memorize scheduling behaviors without fully comprehending their real-world impact.

In many educational settings, instructors attempt to address this gap by manually drawing Gantt charts or stepping through algorithms on a whiteboard. While this can be helpful, it is time-consuming, lacks interactivity, and may still fail to engage all learners. The need for a comprehensive, visually interactive, and analytically rich tool is therefore evident. Such a tool would provide students with a hands-on approach to learning, allowing them to input processes, select algorithms, and observe the resulting schedule and performance metrics.

This project aims to solve the aforementioned challenges by developing a visual and interactive CPU scheduling simulator. The tool will support multiple algorithms—such as FCFS, SJF, Priority Scheduling, and Round Robin—allowing users to see how each performs with identical input data. It will generate real-time Gantt charts that depict process execution in a timeline format, visually highlighting the differences between algorithms. In addition to graphical outputs, the tool will provide performance metrics such as average waiting time, average turnaround time, and individual process statistics.

The simulator will be designed with usability in mind, ensuring that students of all levels can easily interact with it. Its web-based or desktop interface will allow for process customization, algorithm selection, and instant visualization. The goal is to offer an engaging educational experience that encourages experimentation, deepens understanding, and facilitates comparisons between various scheduling strategies.

Moreover, this project supports future extensions such as multi-processor simulations, real-time scheduling policies, and additional performance measurements. These enhancements can further broaden the tool's utility beyond academic use into system analysis and research domains.
By offering a platform for visual and analytical exploration of CPU scheduling, this project addresses a long-standing gap in operating systems education. It empowers learners with a more intuitive and insightful way to master one of the most essential mechanisms in computing systems.

Expected Outcome:

- A web-based simulation tool.
- Visual scheduling output with Gantt charts.
- Dynamic metric calculation.
- Support for multiple CPU scheduling algorithms.

Chapter 2

Background/ Literature Survey

In the field of computer science and engineering, particularly within operating systems, the management of processes plays a central role. Among the most vital aspects of process management is **CPU scheduling**, which is responsible for determining the order in which processes in the ready queue are allocated the processor. This allocation impacts key system performance metrics such as turnaround time, waiting time, throughput, and CPU utilization.

CPU scheduling becomes especially important in multiprogramming and multitasking systems where multiple processes compete for CPU access. The aim of an efficient CPU scheduling mechanism is to optimize the use of the processor by minimizing idle time and ensuring fair and fast execution of all processes. The manner in which this is achieved is governed by scheduling algorithms, each suited to specific types of workloads and performance goals.

Numerous CPU scheduling algorithms have been proposed and analyzed extensively in academic literature. Among these, the most commonly discussed include:

1. **First-Come, First-Served (FCFS):** The simplest form of CPU scheduling, FCFS allocates the CPU to processes in the order of their arrival. While easy to implement and understand, this method suffers from the "convoy effect," where short processes are forced to wait for long ones to finish, resulting in poor average response time and lower overall efficiency.

2. **Shortest Job First (SJF):** SJF selects the process with the smallest execution time. This method minimizes the average turnaround time and is considered optimal under certain conditions. However, it requires precise knowledge of future burst times, which is rarely available in real systems. Furthermore, longer processes may suffer from starvation if short jobs keep arriving.

3. **Priority Scheduling:** Each process is assigned a priority, and the CPU is allocated to the process with the highest priority. While this system allows for control over which tasks are most important, it too can cause starvation if low-priority processes are never selected. Solutions like **aging**—increasing priority the longer a process waits—can mitigate this issue.

4. **Round Robin (RR):** A widely-used method in time-sharing systems, RR assigns a fixed time quantum to each process in a cyclic order. This ensures fairness and prevents starvation. However, its performance heavily depends on the size of the time quantum—too large, and it behaves like FCFS; too small, and overhead from context switching increases.

Each of these algorithms has been studied theoretically and tested in various environments to understand its implications. Researchers such as Silberschatz et al. and Tanenbaum have contributed significantly to the understanding and classification of these algorithms. Their studies provide mathematical models, case analyses, and empirical evaluations which continue to be the backbone of academic courses and system-level implementations.

While foundational texts and classroom lectures form the theoretical base, they often fail to provide a **visual or dynamic representation** of CPU scheduling behavior. To bridge this educational gap, simulation tools have emerged as effective teaching aids. Simulators allow students to input custom process parameters such as arrival time, burst time, and priority.

The tool then visualizes the process execution sequence via Gantt charts and computes performance metrics such as average waiting time and average turnaround time.
One such example is the CPU Scheduling Algorithm Visualiser which has proven to be a valuable resource for learning. This online tool enables users to test different scheduling algorithms by inputting process data and viewing animated execution timelines. It provides a hands-on approach to learning by converting static textbook content into interactive simulations. The platform shows how different algorithms impact scheduling metrics and helps learners intuitively understand the strengths and weaknesses of each strategy.

However, most existing simulators are limited in scope. Some support only a few algorithms or restrict input flexibility. Others lack real-time interactivity or fail to compute detailed statistics. As such, there remains a need for more robust and extensible tools that provide deeper insights into CPU scheduling. Academic literature also suggests that visual, simulation-based learning significantly enhances student engagement and retention in technical subjects such as operating systems.

Additionally, in real-world systems, the choice of CPU scheduling algorithm depends on the specific context—batch systems may prioritize throughput, interactive systems focus on response time, and real-time systems require deterministic behavior. Understanding these nuances is essential for system designers, which further validates the need for detailed simulations and comparative analysis tools.
In conclusion, the study of CPU scheduling is extensive and spans both theoretical and practical dimensions. Through decades of academic exploration, a solid understanding of different scheduling policies has been established. Yet, with evolving system requirements and a growing demand for effective teaching tools, there is a continuous drive to develop new ways to present and explore these algorithms.

Our project builds on this legacy by offering a simulator that not only visualizes CPU scheduling but also fosters analytical thinking through metric evaluation, algorithm comparison, and interactive learning.

Chapter 3

Objectives

The main objective of this project is to design and implement a comprehensive CPU scheduling simulator that helps to visualize and understand the functioning of different CPU scheduling algorithms. The tool will serve as a hands-on educational resource to reinforce theoretical concepts through practical simulations.

The specific objectives of the project are as follows:

1. Simulation of Major Scheduling Algorithms: To implement First-Come-First-Serve (FCFS), Shortest Job First (SJF), Priority Scheduling (both preemptive and non-preemptive), and Round Robin scheduling algorithms.
2. Interactive Input Interface: To create a user interface that allows users to enter process parameters such as burst time, arrival time, and priority, and to select the desired scheduling algorithm.
3. Gantt Chart Visualization: To generate a real-time Gantt chart that visually represents the execution order and time slots of each process.
4. Performance Metric Calculation: To compute and display important scheduling metrics including:
    - Waiting Time (WT)
    - Turnaround Time (TAT)
    - Completion Time (CT)
    - Average WT and TAT
5. Algorithm Comparison: To allow side-by-side comparison of metrics for different algorithms using the same process input.
6. Educational Focus: To provide an intuitive and visually engaging learning experience that supports theoretical understanding and analytical thinking.
7. Scalability and Extensibility: To lay the groundwork for future extensions such as real-time scheduling, multi-core CPU simulation, and interactive animations.

This project aims to contribute meaningfully to the educational tools available in the field of operating systems by providing a robust, visually rich, and analytically accurate simulator that can be used in both academic and self-learning environments.

Chapter 4

Hardware and Software Requirements

1.1 Hardware Requirements

Sl. No	Name of the Hardware	Specification
1.	Processor	Intel Core i5 / AMD Ryzen 5 or higher
2.	RAM	Minimum 4GB
3.	Storage	20GB+

1.2 Software Requirements

Sl. No	Name of the Software	Specification
1.	Programming Language	JavaScript / Python
2.	Framework/Library	ReactJS / CanvasJS / Tkinter
3.	OS	Windows/Linux/MacOS
4.	Browser	Chrome / Firefox

Chapter 5

Possible Approach/ Algorithms

The proposed CPU scheduling simulator will be developed using a structured and modular approach, ensuring both functional clarity and extensibility for future enhancements. The system will be composed of four main components: a user interface for input, an algorithm engine for processing logic, a visualization module for graphical output, and a results calculator for generating key metrics.

5.1 System Architecture

The architecture of the simulator will follow a model-view-controller (MVC) pattern. The model will include data structures for representing processes and their attributes (arrival time, burst time, priority, etc.). The controller will manage user interaction, trigger scheduling logic based on selected algorithms, and pass results to the view. The view will be responsible for dynamically generating Gantt charts, tables, and other visuals.

5.2 Process Input

The simulator will allow users to input a list of processes, with custom attributes such as:

- Process ID
- Arrival Time
- Burst Time
- Priority (if applicable)
- Time Quantum (for Round Robin)

Inputs will be entered via form fields in the GUI. Once submitted, the processes will be passed to the selected scheduling algorithm.

5.3 Scheduling Algorithm Implementation

The simulator will include the following algorithms, implemented with both their logic and visual outputs:

1. First-Come, First-Served (FCFS):

- Non-preemptive.
- Simple linear queue where the first arriving process is scheduled first.
- Ideal for batch systems but results in long average waiting times when longer processes are executed early.

2. Shortest Job First (SJF):

- Both preemptive and non-preemptive versions will be implemented.
- Selects the process with the smallest burst time among available processes.
- Provides optimal turnaround time but risks starvation of long processes.

3. Priority Scheduling:

- Processes are executed based on priority level.

- Both preemptive and non-preemptive approaches.
- Includes optional aging mechanism to prevent indefinite postponement of lower-priority processes.

4. Round Robin (RR):

- Each process is assigned a time quantum and executed cyclically.
- Preemptive in nature and best suited for time-sharing systems.
- Time quantum can be adjusted to demonstrate its effect on performance and context switching overhead.

5.4 Scheduling Visualization

Each algorithm will be accompanied by a real-time Gantt chart:

- Processes will be displayed in colored blocks with labels.
- The Gantt chart will update dynamically based on the selected algorithm.
- Tooltips or labels may show key values like start time, end time, and process ID.

Additionally, a tabular section will display:

- Waiting Time per process
- Turnaround Time per process
- Completion Time
- Average Waiting Time
- Average Turnaround Time

These metrics will help users compare the practical effectiveness of each algorithm based on their custom input.

5.5 User Experience

The interface will be designed with simplicity and educational value in mind:

- Dropdown menus for algorithm selection.
- Dynamic tables for input and output.
- Buttons to reset, clear input, and compare outputs.
- Support for exporting results as images or downloadable files (optional).

5.6 Comparison Engine

A unique feature of the simulator will be the side-by-side comparison engine:

- Allows running all algorithms on the same input.
- Displays comparative metrics in one view.
- Enables students to observe the trade-offs between algorithms visually and analytically.

Chapter 6

Conclusion

CPU scheduling is an essential and foundational component of operating systems, enabling efficient and fair allocation of processor time among competing processes. In the realm of academic learning, while theoretical understanding is imparted through books and lectures, practical and interactive comprehension often lags behind. This project bridges that gap by offering an immersive, dynamic, and educational tool that brings CPU scheduling algorithms to life.

The development of an interactive simulator for CPU scheduling provides a crucial learning aid to students, educators, and software engineers. Through simulation, users are able to visualize how different scheduling algorithms function, how they affect system performance, and how to interpret critical performance metrics such as turnaround time, waiting time, and response time. The use of Gantt charts as a visual representation allows learners to intuitively grasp the time-sequenced behavior of algorithms, while dynamic tables present quantitative feedback that reinforces the learning process. By implementing algorithms such as FCFS, SJF, Priority Scheduling, and Round Robin, the simulator covers a comprehensive range of classical scheduling policies, each suited to specific real-world scenarios. The inclusion of both preemptive and non-preemptive variations allows users to understand context switching and CPU sharing from multiple angles. The ability to configure parameters such as burst time, arrival time, priority, and quantum size further enriches the interactive experience and encourages experimentation.

Another key takeaway from the project is its contribution to digital pedagogy. As the world shifts toward e-learning and virtual labs, simulation-based tools like this one have become integral to curriculum design. They support active learning and promote critical thinking by encouraging users to test hypotheses, analyze results, and refine their understanding. This simulator not only reinforces conceptual learning but also fosters curiosity and deeper inquiry.

In conclusion, this CPU scheduling simulator is not just a technical project but a step forward in modernizing operating systems education. It empowers learners to engage with abstract concepts in a tangible way, demystifies the inner workings of process scheduling, and nurtures the skills required for systems-level thinking. Whether used in classrooms, labs, or personal study environments, this project has the potential to significantly improve the learning curve and engagement in the field of operating systems.

References

[1] Silberschatz, Abraham, Peter B. Galvin, and Greg Gagne. "Operating System Concepts," 10th Edition, Wiley, 2018.

[2] Tanenbaum, Andrew S., and Herbert Bos. "Modern Operating Systems," 4th Edition, Pearson, 2014.

[3] William Stallings. "Operating Systems: Internals and Design Principles," 9th Edition, Pearson, 2017.

[4] Dhamdhere, D. M. "Operating Systems: A Concept-Based Approach," McGraw Hill, 3rd Edition, 2012.

[5] GeeksforGeeks. "CPU Scheduling in Operating Systems." https://www.geeksforgeeks.org/cpu-scheduling-in-operating-systems