Process Scheduler Simulator

Submitted By

Student Name	Student ID
Md. Mehedi Hasan	0242220005101432
Sumaiya Akter	0242310005101149
Forhad Alam Sourav	0242310005101853
Abdullah Bin Arif Apon	0242310005101850

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE324 Operating Systems Lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

06 December 2025

DECLARATION

We hereby declare that this lab project titled **“Process Scheduler Simulator”** has been completed by our group under the supervision of **Md. Jahangir Alam**, Lecturer, Department of Computer Science and Engineering, Daffodil International University. We confirm that this work is original and has not been submitted elsewhere for academic grading.

Submitted To:

Md. Jahangir Alam
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

Md. Mehedi Hasan 0242220005101432 Dept. of CSE, DIU	**Sumaiya Akter** 0242310005101149 Dept. of CSE, DIU
Forhad Alam Sourav 0242310005101853 Dept. of CSE, DIU	**Abdullah Bin Arif Apon** 0242310005101850 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:.

Table 1: Course Outcome Statements

CO's	Statements
CO1	Explain the fundamental concepts of CPU scheduling in operating systems.
CO2	Apply classical scheduling algorithms to solve real scheduling problems
CO3	Analyze performance metrics like waiting time and turnaround time.
CO4	Implement and evaluate scheduling algorithms using C programming.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Mapping of CO → PO

- **CO1 → PO1:** Understanding OS scheduling foundations.
- **CO2 → PO2:** Applying scheduling algorithms computationally.
- **CO3 → PO3:** Analyzing execution patterns and evaluating results.
- **CO4 → PO3:** Developing a simulator as a real-world problem-solving tool.

Table of Contents

Chapter 1: Introduction

This chapter introduces the project background, the need for CPU scheduling, motivation, objectives, and expected outcome.

1.1 Introduction

An operating system manages multiple processes by allocating CPU resources efficiently. CPU scheduling determines the order in which processes run. This project implements a **Process Scheduler Simulator** that supports four major scheduling algorithms: **FCFS, SJF, Round Robin (RR), and Priority Scheduling**. The simulator demonstrates how different algorithms affect waiting time and execution order.

1.2 Motivation

CPU scheduling is one of the core functions of an operating system. Students often find it difficult to visualize how processes are executed internally. Therefore, a simulation tool helps understand:

- How processes are queued
- How CPU time is distributed
- How waiting time changes
- Why some algorithms perform better than others

Additionally, developing a simulator strengthens programming skills and OS concepts.

1.3 Objectives

The main objectives of this project are:

1. To study and implement classical CPU scheduling algorithms.
2. To simulate real-time scheduling behavior using C.
3. To compute and analyze average waiting time for each algorithm.
4. To visualize execution sequences.
5. To compare algorithms based on performance.

1.4 Feasibility Study

The feasibility of the project is analyzed in terms of:

- **Technical Feasibility:** Uses only a C compiler; no extra hardware needed.
- **Operational Feasibility:** Students can operate the CLI interface easily.
- **Economic Feasibility:** No cost associated; open-source environment.

- **Time Feasibility:** Implementation and testing can be done within 1 week.

Existing tools (e.g., Gantt chart generators, web simulators) are complex. Our project offers a lightweight, offline, and beginner-friendly option.

1.5 Gap Analysis

Existing schedulers often:

- Lack transparency in showing step-by-step execution
- Do not allow user-defined input
- Focus only on visualization but not algorithmic understanding

Our simulator fills these gaps by providing:

- Full control over arrival and burst time inputs
- Real-time sequence simulation
- Clear display of waiting time and algorithm behavior

1.6 Project Outcome

The project produces a functional console-based scheduler tool capable of:

- Running FCFS, SJF, RR, and Priority Scheduling
- Displaying execution sequences
- Calculating average waiting times
- Allowing repeated experiments

Chapter 2: Proposed Methodology / Architecture

This chapter explains how the system works, its workflow, and design architecture.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The simulator consists of:

Input Module: Takes process arrival time, burst time, and priority.

Scheduler Module: Implements four scheduling algorithms.

Output Module: Displays waiting time, execution order, and summary.

2.1.2 System Design

System Workflow:

1. User selects a scheduling algorithm.
2. User enters process information.
3. Algorithm simulates scheduling.
4. Results are displayed.

Scheduling Algorithms Implemented:

- **FCFS:** Simple queue-based scheduling.
- **SJF:** Shortest job first among available processes.
- **RR:** Time-sliced CPU allocation.
- **Priority:** Highest priority executed first.

2.1.3 UI Design

```
========================================
PROCESS SCHEDULER SIMULATOR
========================================
1. First Come First Serve
2. Shortest Job First
3. Round Robin
4. Priority Scheduling
0. Exit
========================================
```

2.2 Overall Project Plan

- Step 1: Study scheduling algorithms
- Step 2: Write C code modules
- Step 3: Integrate all algorithms into menu system
- Step 4: Test multiple input cases
- Step 5: Document result

Phase 1 – Requirement Analysis (Day 1)

- Identify the scheduling algorithms required: FCFS, SJF, RR, Priority.
- Understand input/output requirements.
- Review OS theory for algorithm design.
- Determine feasibility of implementation in standard C.

Phase 2 – System Design (Day 2)

- Design input module for dynamic process handling.
- Define variables: arrival time, burst time, priority, waiting time.
- Plan scheduling logic for all four algorithms.
- Prepare menu-driven UI layout.
- Draft pseudocode for each scheduler.

Phase 3 – Implementation (Day 3–4)

- Write modular functions: `fcfs()`, `sjf()`, `rr()`, `priority_sch()`.
- Implement time progression logic (timestep simulation).
- Implement sequence recording for execution order.
- Add result printing (waiting time, sequence, average waiting time).
- Integrate functions into main menu.

Phase 4 – Testing & Debugging (Day 5)

- Prepare sample test cases with different arrival, burst, and priority values.
- Test edge cases:
    - identical arrival times
    - large burst time differences
    - high/low priority variations
    - empty CPU time slots
- Fix issues related to time increment, RR quantum handling, and waiting time calculation.

Phase 5 – Performance Evaluation (Day 6)

- Compare algorithm outputs.
- Measure differences in average waiting time.
- Analyze which algorithms perform better for specific scenarios.

Chapter 3: Implementation and Results

This chapter details how the simulator was built using C and presents results.

3.1 Implementation

Implementation highlights:

- Used arrays to store arrival, burst, priority, waiting time.
- Used loops to simulate CPU time passing.
- Round Robin uses quantum-based loop for fairness.
- SJF and Priority use selection logic to pick optimal processes.
- Final output includes waiting time, finishing time, and sequence.

3.2 Performance Analysis

Performance is evaluated using:

- **CPU Waiting Time**
- **Execution Sequence**
- **Algorithm Behavior**

Observations:

- FCFS performs poorly with long jobs arriving early.
- SJF gives lowest waiting time for short jobs.
- RR gives fairness but may increase waiting time.
- Priority scheduling depends heavily on assigned priorities.

3.3 Results and Discussion

Each algorithm produces a different order of execution. The simulator confirms theoretical scheduling behavior. The project successfully demonstrates how small changes in arrival, burst, or priority values affect CPU scheduling.

Chapter 4: Engineering Standards and Mapping

This chapter connects the engineering aspects of the project with ethical, environmental, and teamwork considerations.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

Efficient scheduling improves device responsiveness and application load time.

4.1.2 Impact on Society & Environment

Optimized CPU scheduling reduces energy use → Less heat → Longer system lifespan.

4.1.3 Ethical Aspects

- No biased scheduling logic
- Transparent, open-source approach
- Responsible coding practices

4.1.4 Sustainability Plan

The simulator can be upgraded to support:

- GUI
- Preemptive algorithms
- Real-time scheduling

4.2 Project Management and Team Work

Teamwork distribution:

- Research: ALL members
- Coding: Mehedi, Sourav, Apon
- Testing: Sumaiya
- Documentation: Full group

4.3 Complex Engineering Problem Mapping

Program Outcome Mapping

- **PO1:** Understanding algorithms → Achieved
- **PO2:** Applying concepts in C → Achieved
- **PO3:** Analyzing results → Achieved

Complex Problem Solving

- Involves structured algorithm design
- Requires analyzing multiple constraints (arrival, burst, priority)

Engineering Activities

- Creating a functional system
- Testing correctness
- Evaluating performance impacts

4.3.1 Mapping of Program Outcome (POs)

The Process Scheduler Simulator aligns with the following Program Outcomes:

PO	Description	Justification of Attainment
PO1	Engineering Knowledge	Students apply OS concepts, scheduling theory, and algorithm design to create a functional simulator.
PO2	Problem Analysis	The project requires analyzing scheduling problems, identifying constraints (arrival time, burst time), and selecting appropriate scheduling models.
PO3	Design/Development of Solutions	Students design and implement four complete scheduling algorithms and integrate them into a working system.
PO4	Investigation & Interpretation	Performance results such as average waiting time are interpreted and compared across scheduling strategies.

Justification Summary:
The project requires understanding complex OS behavior, analyzing algorithm characteristics, designing modular code, and interpreting results—covering a wide range of engineering competencies.

Chapter 5: Conclusion

This chapter summarizes the project, identifies limitations, and proposes future improvements.

5.1 Summary

The Process Scheduler Simulator successfully implements and demonstrates four major scheduling techniques. It is accurate, user-friendly, and educational.

5.2 Limitation

- No preemptive SJF or Priority
- No graphical Gantt chart
- Single-core simulation only

5.3 Future Work

- Add GUI
- Add preemptive priority/SJF
- Add multi-core scheduling
- Add real-time scheduling (EDF, RM)

References

1. Silberschatz, Abraham. "Operating System Concepts."
2. Tanenbaum, Andrew. "Modern Operating Systems."
3. OS Project Source Code (C Implementation)