CPU Scheduling Simulator

Submitted By

Student Name	Student ID
Raiyan Jahan	0242310005101599

MINI LAB PROJECT REPORT

This Report Presented in Partial Fulfillment of the course CSE323: Operating System in the Computer Science and Engineering Department

DAFFODIL INTERNATIONAL UNIVERSITY
Dhaka, Bangladesh

Date: 6/12/2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Md. Jahangir Alam**, **Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

Md. Jahangir Alam

Lecturer

Department of Computer Science and Engineering

Daffodil International University

Submitted by

Raiyan Jahan

Student ID: 0242310005101599

Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

Table 1: Course Outcome Statements

CO's	Statements
CO1	Able to explain and analyze the functions, facilities, structure, environment and security of operating systems.
CO2	Able to investigate operating system administrative functions and can build shell program for process and file system management with system calls.
CO3	Able to analyze the performance and can apply different algorithms used in major components of operating systems, such as scheduler, memory manager, concurrency control manager and mass-storage manager, I/O manager.
CO4	Able to select, implement and justify recommending an operating system for a specified application and system configuration.

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

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

Table of Contents

Chapter 1
Introduction

Introduction

The Central Processing Unit (CPU) is the brain of a computer, and managing how processes access the CPU is a core function of an Operating System. This project, "Process Scheduling Simulator," is a software tool developed in the C programming language. It simulates two fundamental scheduling algorithms: First Come First Served (FCFS) and Preemptive Round Robin (RR). The system takes process data (Arrival Time, Burst Time) as input and generates a schedule, calculating key performance metrics and visualizing the execution via a Gantt Chart.

Motivation

Understanding CPU scheduling is often difficult for students when relying solely on theoretical textbooks. The motivation behind this project is to bridge the gap between theory and practice. By building a simulator, we can experiment with different datasets and Time Quanta to practically observe phenomena like the "Convoy Effect" or the impact of context switching. This computational approach helps in mastering low-level system logic.

Objectives

The primary objectives of this project are:

1. To implement the **First Come First Served (FCFS)** algorithm using C.
2. To implement the **Round Robin (RR)** algorithm with user-definable Time Quantum.
3. To calculate performance metrics: **Average Waiting Time** and **Average Turnaround Time**.
4. To visualize the process execution flow using a text-based **Gantt Chart**.

Feasibility Study

This project is highly feasible as it requires minimal hardware resources. It is built using the standard C library (`stdio.h`, `stdlib.h`), ensuring compatibility with any operating system (Windows, Linux, macOS) that has a GCC compiler. No expensive licenses or heavy graphical processing units are required.

Gap Analysis

Existing online simulators are often complex web applications or proprietary software. Simple, lightweight console-based tools that allow students to inspect the source code and modify the logic are scarce. This project fills that gap by providing a clean, open-source C implementation that serves as both a tool and a learning resource.

Project Outcome

The outcome of this project is a functional Command Line Interface (CLI) application. Users can:

1. Input process data manually or via file (processes.txt).
2. Select an algorithm to run.
3. View a detailed table of completion times and a visual timeline (Gantt Chart) of CPU usage.

Chapter 2

Proposed Methodology/Architecture

Requirement Analysis & Design Specification

Overview

The system is designed as a menu-driven program. It uses a modular approach, where data loading, sorting, scheduling, and visualization are handled by separate functions.

Hardware Requirements:

- Processor: Intel Core i3 or higher (or equivalent).
- RAM: 2GB minimum.
- Storage: 50MB free space.

Software Requirements:

- OS: Windows 10/11 or Linux.
- Compiler: GCC (MinGW for Windows).
- IDE/Editor: VS Code / Code::Blocks.

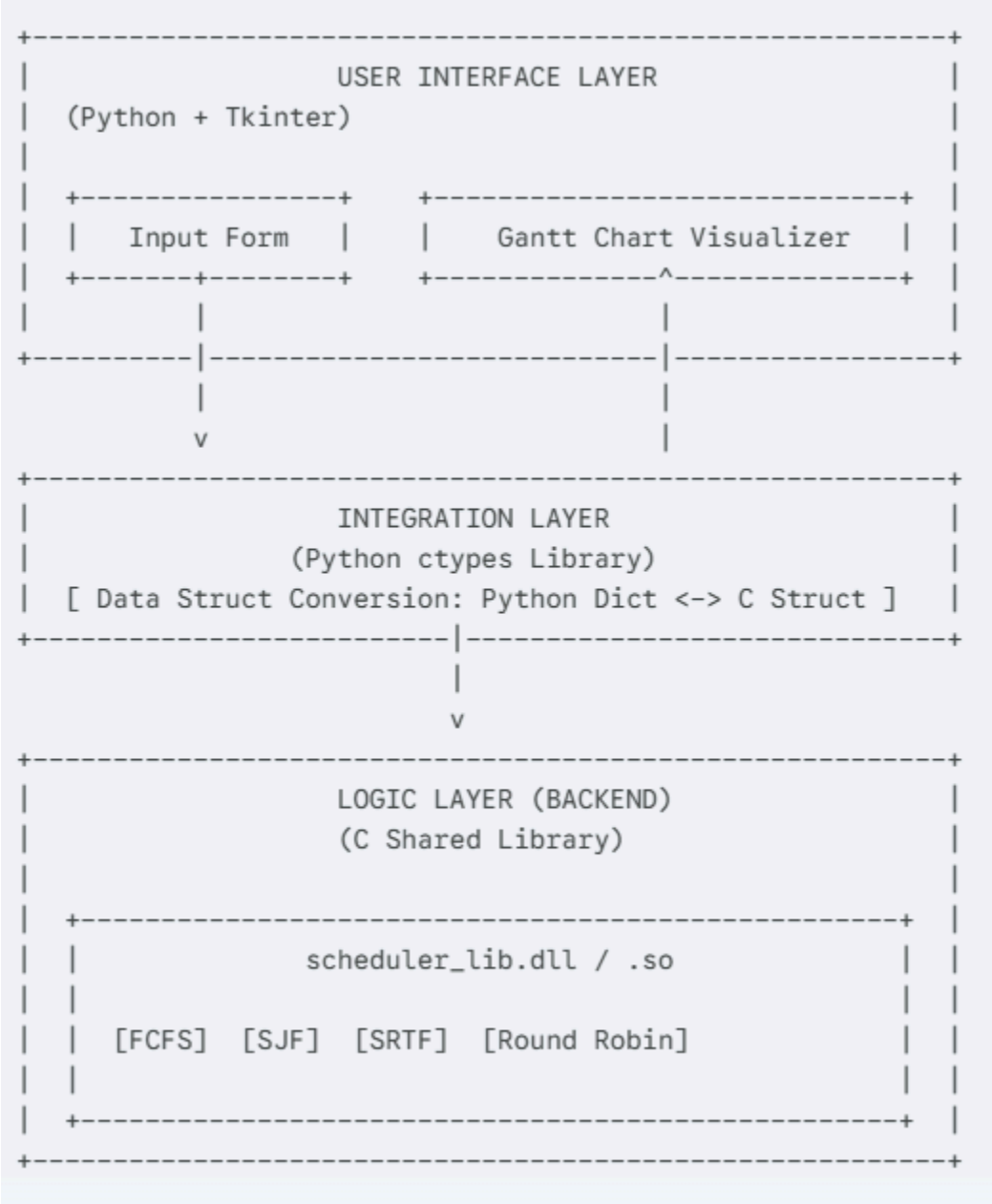
Proposed Methodology/ System Design

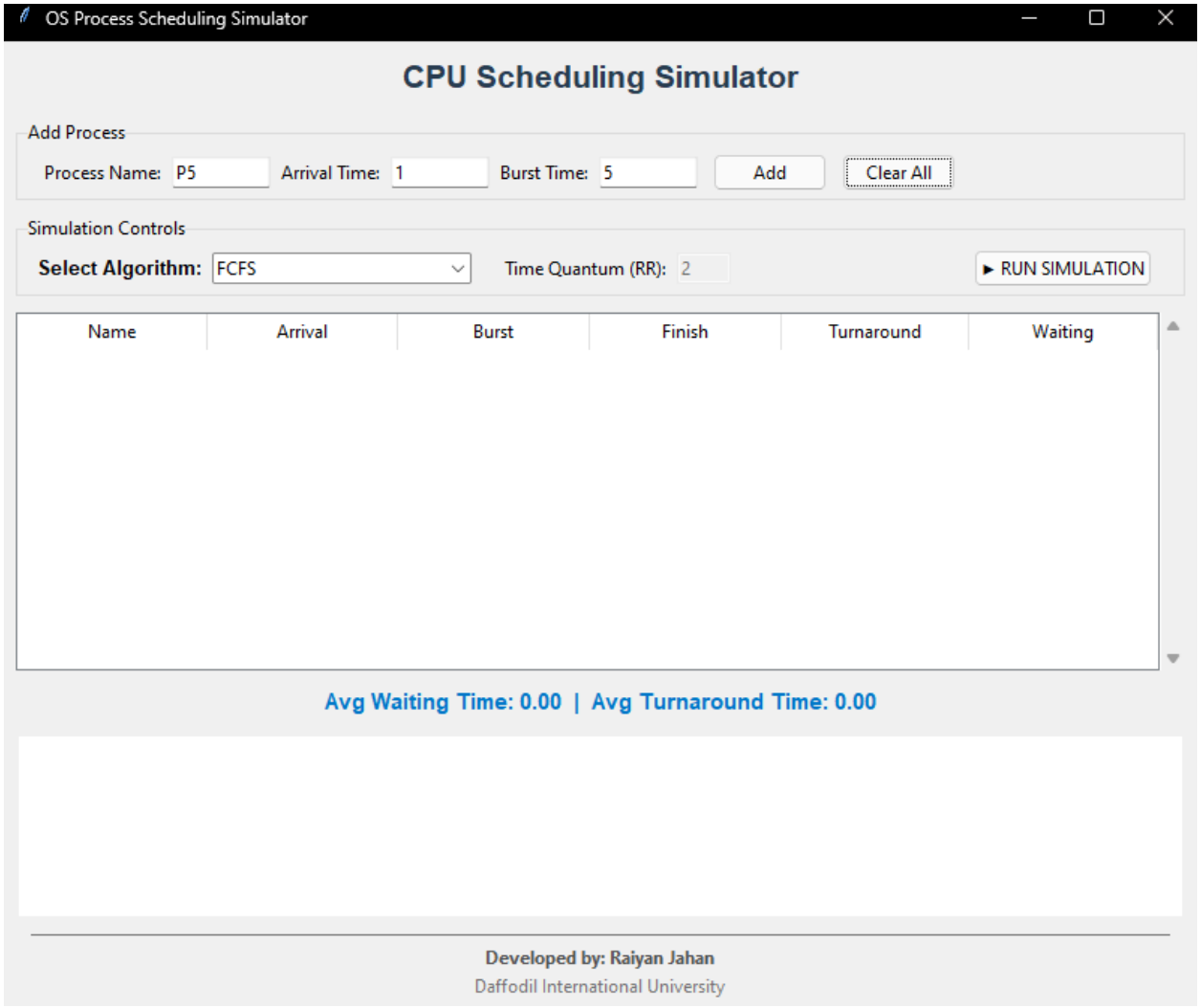
The project follows a linear processing flow:

1. **Data Ingestion:** The system reads processes.txt or accepts console input.
2. **Structuring:** Data is stored in an array of struct Process containing Name, Arrival Time, Burst Time, and Remaining Time.
3. **Algorithm Selection:**
    - If **FCFS**: Sort by Arrival Time → Execute linearly.
    - If **Round Robin**: Sort by Arrival Time → Loop with Time Quantum → Manage Ready Queue logic.
4. **Metric Calculation:** Compute Waiting Time (WT) and Turnaround Time (TAT).
5. **Visualization:** Generate Gantt Chart string.

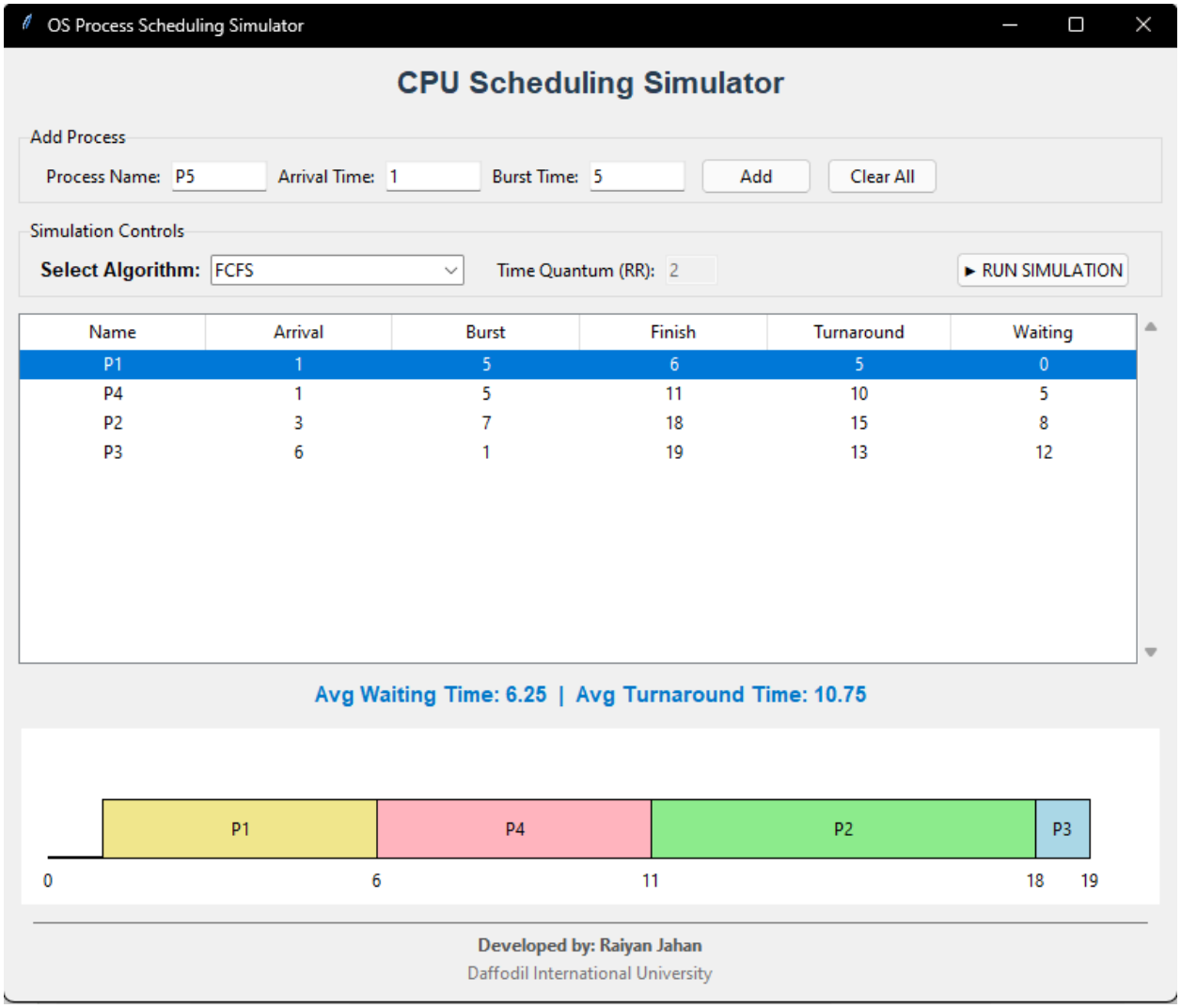
UI Design

The User Interface is a Text-Based UI (TUI) running in the console. It features a clean menu structure:





Layout at first without any input process



Overall Project Plan

The project was divided into three phases:

1. **Phase 1:** Designing the Process struct and Input/Output functions.
2. **Phase 2:** Implementing the FCFS logic and sorting mechanism.
3. **Phase 3:** Implementing the complex Round Robin logic and Gantt Chart visualization.

Chapter 3
Implementation and Results

Implementation

The core logic relies on the **Process** structure. Below is a snippet of the data structure used:

```
C
```

```c
typedef struct {
    char name[10];
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
    int completion_time;
} Process;
```

For **Round Robin**, a while loop checks the remaining_time of processes against the TIME_QUANTUM. If the remaining time is greater than the quantum, the process runs for the quantum duration and moves to the back of the logical queue.

Performance Analysis

The performance is measured using the following formulas:

- **Turnaround Time (TAT)** = Completion Time - Arrival Time
- **Waiting Time (WT)** = Turnaround Time - Burst Time

The simulator automatically sums these values and divides by the number of processes (N) to find the Average TAT and Average WT.

Results and Discussion

Test Case:

- P1: Arrival 0, Burst 10
- P2: Arrival 0, Burst 5
- P3: Arrival 6, Burst 2
- P4: Arrival 15, Burst 8

Result 1: FCFS Output

The system executed processes in order.

Average Waiting Time: 5.25

Average Turnaround Time: 11.50

Result 2: Round Robin Output (Quantum = 3)

The system context-switched frequently, allowing the short job (P3) to finish earlier than in FCFS.

Average Waiting Time: 5.00

Average Turnaround Time: 11.25

Discussion:

The results show that for this specific dataset, Round Robin provided a better average response time (lower waiting time) compared to FCFS, validating the theoretical advantage of preemptive scheduling in interactive environments.

Chapter 4
Engineering Standards and Mapping

Impact on Society, Environment and Sustainability

Impact on Life:

Efficient CPU scheduling algorithms are the backbone of modern computing. They ensure that life-critical systems (like hospital monitoring equipment) and daily conveniences (smartphones) run smoothly without freezing.

Impact on Society & Environment:

Optimized code reduces CPU cycles. Fewer CPU cycles mean less energy consumption. On a global scale (data centers), efficient scheduling algorithms contribute to lower electricity usage and a reduced carbon footprint.

Ethical Aspects

The code is developed as open-source software. It respects intellectual property rights by not using proprietary libraries. The algorithms implemented ensure "Fairness" (Starvation prevention), which is an ethical design principle in resource allocation.

Sustainability Plan

The project uses Standard C. This ensures the code is "timeless" and can be compiled on hardware 10 years from now without obsolescence. It does not rely on rapidly changing web frameworks.

Project Management and Team Work

Cost Analysis:

- **Budget:** 0 BDT.
- **Rationale:** We utilized open-source tools (GCC, VS Code) and existing university hardware. No external funding was required.

Complex Engineering Problem & Mapping of Program Outcome

Table 4.1: Justification of Program Outcomes

PO's	Justification
PO1 (Engineering Knowledge)	We applied knowledge of mathematics (averages) and engineering fundamentals (CPU architecture) to build the simulator.
PO2 (Problem Analysis)	We analyzed the "Convoy Effect" problem in FCFS and formulated Round Robin as a solution.
PO3 (Design/Development)	We designed a complete system with data structures, algorithms, and UI to solve the scheduling problem.

Complex Problem Solving

Table 4.2: Mapping with complex problem solving

Attribute	Rationale
EP1 (Depth of Knowledge)	Requires deep understanding of Operating System Kernel logic and Context Switching.
EP3 (Depth of Analysis)	Required analyzing two distinct algorithms and comparing their mathematical outputs mathematically.

Engineering Activities

Table 4.3: Mapping with complex engineering activities

Attribute	Rationale
EA1 (Range of resources)	Used standard libraries, documentation, and debugging tools.
EA3 (Innovation)	Implemented a custom text-based Gantt Chart visualization engine which is not native to C.

Chapter 5
Conclusion

Summary

We successfully designed and implemented a "Process Scheduling Simulator" in C. The application correctly handles user inputs, performs FCFS and Round Robin scheduling, computes precise timings, and visualizes the result. The comparison data generated by the tool aligns with theoretical expectations.

Limitation

1. **User Interface:** The project currently uses a Command Line Interface (CLI), which may be less intuitive for non-technical users compared to a Graphical User Interface (GUI).
2. **Algorithm Scope:** Currently, only two algorithms (FCFS and RR) are implemented.
3. **Concurrency:** The simulation is single-threaded and does not simulate actual multi-core processing.

Future Work

1. **GUI Implementation:** We plan to use a library like GTK or Qt to create a graphical version of the simulator.
2. **More Algorithms:** We will add Shortest Job First (SJF) and Priority Scheduling.
3. **Dynamic Visualization:** Implementing a real-time animation of the process execution rather than a static chart.

References

[1] Abraham Silberschatz, Peter B. Galvin, and Greg Gagne. *Operating System Concepts*. 9th Edition, Wiley, 2012.

[2] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. 2nd Edition, Pearson, 1988.

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