|  |  |  |
| --- | --- | --- |
| A logo of a university of sciences  Description automatically generatedA red circle with white text and a book and lightning bolt  Description automatically generated  **MILITARY COLLEGE OF SIGNALS , NUST** | | |
| Complex Engineering Problem  Report | | |
| Interactive CPU Scheduling Simulator | | |
| Submitted To | Dr. Nida Adnan | |
| Subject | Operating Systems | |
| Course | BESE 29 | |
| Section | B | |
| Group Members | | |
| * Zoiba Tabassum * Jannat Shaheer * Izhan Zahoor | | 478562  472248  486865 |

**Introduction**

This project is an interactive CPU Scheduling Simulator designed to help users understand how different CPU scheduling algorithms manage processes in a multitasking system.  
Built using Python and PyQt5, the simulator offers real-time visualization, dynamic Gantt charts, light/dark theme support, pause/reset/resume controls and intelligent algorithm suggestions based on system load and task types.

**Objective**

* To simulate major CPU scheduling algorithms interactively.
* To visualize process execution through a dynamic Gantt chart.
* To calculate and display important performance metrics like Waiting Time, Turnaround Time, CPU Utilization.
* To suggest optimal algorithms based on system conditions.
* To offer users professional GUI experience with theme switching.

**Technology Used**

|  |  |
| --- | --- |
| Tool/Technology | Description |
| Python 3.11 | Programming Language |
| PyQt5 | GUI Development |
| Qt Style Sheets (QSS) | Styling the GUI (like CSS) |
| QTimer | Dynamic Simulation Timer |

**Project Architecture**

* main.py # Application Entry Point
* gui.py # Main GUI Window (PyQt5 Widgets)
* scheduler.py # All scheduling algorithms (FCFS, SJF, Priority, RR, MLFQ)
* process.py # Process class definition
* styles.qss # Light mode styling
* styles\_dark.qss # Dark mode styling

**Features Implemented**

* Add, Edit, Delete Processes.
* Choose between multiple Scheduling Algorithms.
* Real-time Dynamic Gantt Chart simulation.
* Light Mode and Dark Mode switching.
* Pause ,reset, and resume controls
* Full Performance Metrics calculation post-simulation.
* Intelligent Algorithm Suggestions based on task types.
* Professional GUI styling using external .qss files.

**Scheduling Algorithms Implemented**

The simulator includes five major CPU scheduling algorithms, each with distinct characteristics and use cases.

* **First Come First Serve (FCFS)**

First Come First Serve (FCFS) is the simplest and most straightforward scheduling algorithm.  
In FCFS, processes are scheduled strictly in the order in which they arrive in the ready queue.  
The process that arrives first gets executed first without any preemption, meaning that once a process starts executing, it runs to completion before the CPU moves to the next process.  
While FCFS is very simple to implement and fair in terms of arrival times, it can lead to issues such as the "convoy effect," where shorter processes must wait for longer processes to finish, resulting in increased average waiting time and decreased CPU efficiency.  
Despite its drawbacks, FCFS is still widely used in simple batch processing systems.

* **Shortest Job First (SJF) — Non-Preemptive**

Shortest Job First (SJF) scheduling is a strategy where the process with the shortest estimated CPU burst time is selected for execution next. This algorithm minimizes the average waiting time for a set of processes, making it one of the most efficient scheduling methods in terms of throughput and resource utilization. In non-preemptive SJF, once a process starts execution, it cannot be interrupted, even if a shorter process arrives during its execution. The major drawback of SJF is the potential for process starvation, where longer tasks might be delayed indefinitely if short processes continue arriving.  
Accurate prediction of burst time is critical for SJF to work effectively, which can be challenging in dynamic systems.

* **Priority Scheduling — Non-Preemptive**

Priority Scheduling assigns a priority to each process, and the CPU is allocated to the process with the highest priority (lowest numerical value). In non-preemptive priority scheduling, once a process starts executing, it cannot be interrupted even if a higher priority process arrives. This method is useful in environments where some tasks are more important and must be executed ahead of others, such as handling real-time system processes or urgent requests. However, a significant drawback is the possibility of "starvation" — lower-priority processes might wait indefinitely if higher-priority tasks keep arriving. To combat this, techniques like "aging" (gradually increasing the priority of waiting processes) can be used.

* **Round Robin (RR)**

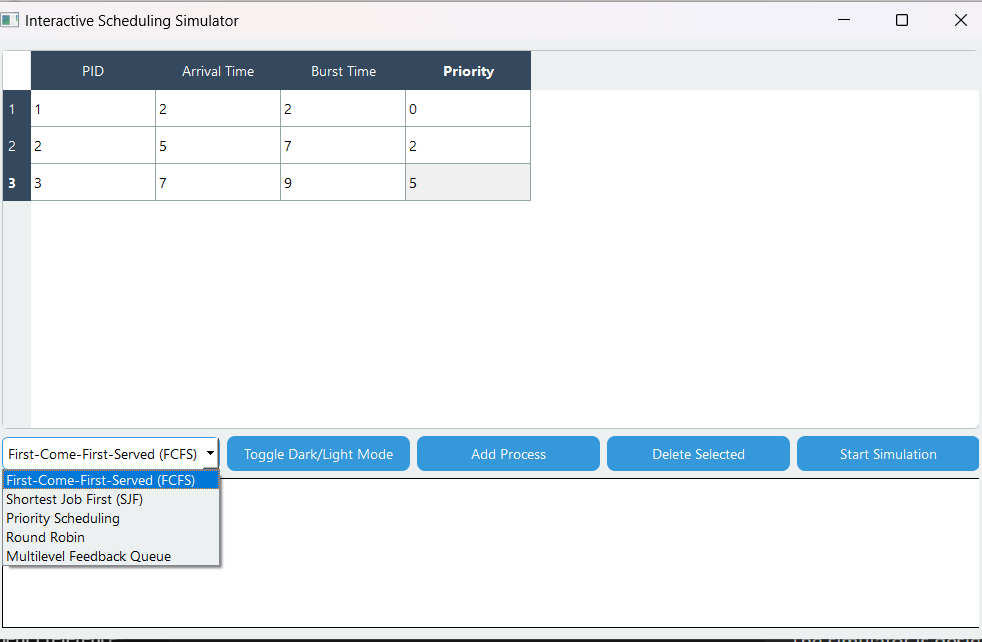
Round Robin (RR) is a preemptive scheduling algorithm specifically designed for time-sharing systems. Each process is assigned a fixed time slice (time quantum) and is allowed to run for that duration. If the process completes its execution within the time quantum, it exits the system; otherwise, it is preempted and placed at the back of the ready queue. This method ensures fairness among all processes and prevents any single process from monopolizing the CPU. Choosing an appropriate time quantum is critical: if the quantum is too large, RR behaves like FCFS; if too small, it results in excessive context switching, reducing CPU efficiency. Round Robin is ideal for systems requiring responsive multitasking, such as interactive user environments.

* **Multilevel Feedback Queue (MLFQ)**

The Multilevel Feedback Queue (MLFQ) is a highly sophisticated scheduling algorithm designed to optimize both short and long processes dynamically. MLFQ uses multiple queues, each with different priority levels and time quantums. New processes typically start in the highest priority queue and are assigned a small time quantum. If they consume their time slice without completing, they are moved down to a lower-priority queue, allowing shorter or interactive processes to be prioritized. This dynamic adjustment between queues enables MLFQ to efficiently handle a mix of CPU-bound and I/O-bound processes. Unlike static algorithms, MLFQ adapts to changing system behavior and balances throughput, turnaround time, and responsiveness. However, MLFQ requires careful configuration of queues, quantum sizes, and promotion/demotion policies to avoid issues like starvation.

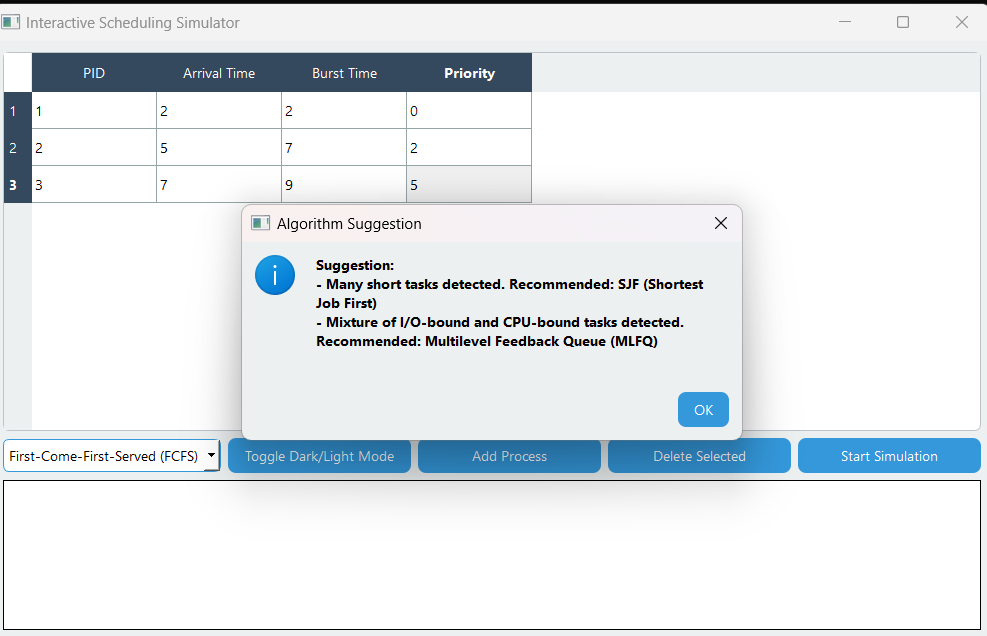
**GUI Design and Dynamic Gantt Chart**

The graphical user interface (GUI) is designed to be both intuitive and functional. The main window displays a table where users can input and manage processes, including setting process details such as arrival time, burst time, and priority. A dropdown menu allows users to select the desired scheduling algorithm. As the simulation runs, the Gantt chart updates dynamically, providing a visual representation of how each process is executed over time. The Gantt chart builds process execution block-by-block, updating in real-time to reflect the chosen scheduling algorithm’s behavior. The design includes **light and dark themes**, which users can toggle for a personalized experience.

**

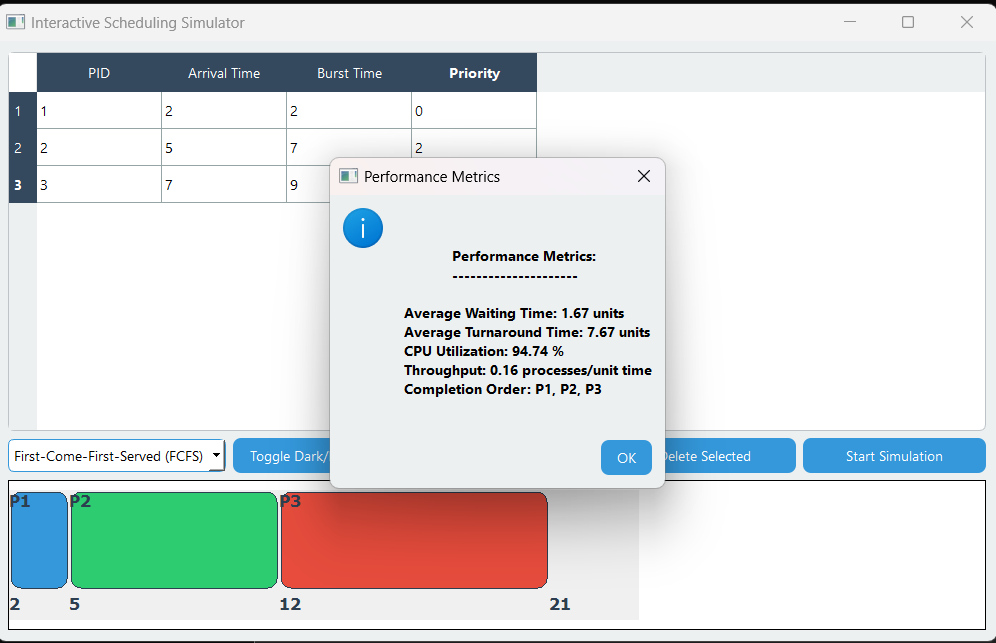
**Intelligent Scheduling Suggestion System**

An advanced feature of this simulator is the **intelligent scheduling suggestion system**, which analyzes the process list before execution and recommends the most appropriate algorithm based on the current system conditions. For example, if there are many short processes, the system might recommend **Shortest Job First (SJF)**, while for systems with long-running interactive tasks, **Priority Scheduling** may be more suitable. The system dynamically adjusts its recommendations, offering suggestions that align with the user’s current process setup, optimizing scheduling decisions and improving system performance.

**

**Performance Metrics**

After each simulation, the **Performance Metrics** window provides critical insights into the system’s performance. Metrics such as **average waiting time**, **average turnaround time**, **CPU utilization**, and **throughput** are calculated to help users evaluate the effectiveness of the chosen scheduling algorithm. These metrics are essential for understanding the trade-offs involved in different scheduling strategies and allow users to compare the performance of various algorithms in a controlled, measurable environment. The results are displayed in a clean and concise manner, making it easy to interpret and analyze the data.

**

**Code Appendix and Outputs**

These outputs not only demonstrate the features of the simulator but also serve as visual evidence of its effectiveness in simulating and visualizing CPU scheduling algorithms.

**Output**

|  |  |
| --- | --- |
| Main GUI WINDOW |  |
| Adding Process |  |
| Selecting Algorithm |  |
| Gant Chart |  |
| Algorithm Suggestion |  |
| Performance metrics |  |
| Dark Mode |  |
| Pause/Resume/ Reset |  |

**Code**

|  |  |
| --- | --- |
| First Come First Serve (FCFS) |  |
| Shortest job first (SJF) |  |
| Priority Queue |  |
| Round Robin |  |
| Multi Level Feedback Queue (MLFQ) |  |
| Add/delete |  |
| Performance Metrics |  |
| Gant Chart |  |
| Perfomance Metrics |  |
| Pause/Reset/Resume |  |

**Conclusion**

In conclusion, the **Interactive CPU Scheduling Simulator** is a comprehensive and educational tool for learning and visualizing CPU scheduling algorithms. Built using Python and PyQt5, the simulator offers a user-friendly interface, real-time process visualization, and intelligent scheduling suggestions to optimize the user experience. The implementation of five major scheduling algorithms, dynamic Gantt chart animations, and performance metrics calculation provides valuable insights into the inner workings of operating systems. This project successfully meets its objectives, offering both educational value and practical utility for students and professionals alike. The project’s modular structure and professional-grade design ensure that it is not only functional but also scalable for future enhancements.