CPU Scheduling

Subject: Operating Systems

Team members:

1	Ahmed Youssef Yousef mohamed	2201789
2	Marwan Yaser Elserafy	2202152
3	Omar Ahmed Mohamed	2201944
4	Ziad Taha Mohamed	2201600

Index of content

Introduction

This project demonstrates how CPU scheduling algorithms work in operating systems. CPU scheduling determines the order in which processes are executed to improve system performance and efficiency.

The project simulates four main algorithms:

- **FCFS (First Come, First Served):** Executes processes in the order they arrive.
- **SJF (Shortest Job First):** Executes the shortest process first.
- **Round Robin (RR):** Each process runs for a fixed time slice (quantum) in a cycle.
- **Priority Scheduling:** Executes processes based on their priority.

By visualizing these algorithms, the project helps understand their logic and impact on system performance.

Objectives

The main objectives of this project are:

- **Understand CPU Scheduling Algorithms:** To learn how different scheduling algorithms work and their role in process management.
- **Implement and Compare Algorithms:** To apply various scheduling algorithms (FCFS, SJF, Round Robin, and Priority) and analyze their performance.
- **Visualize Results:** To present the scheduling process and outcomes through a clear and interactive user interface.

System Overview (Inputs)

The system requires the following inputs to schedule and execute processes:

- **Process ID:** A unique identifier for each process, either assigned manually by the user or generated automatically.
- **Arrival Time:** A unique identifier for each process, either assigned manually by the user or generated automatically.
- **Burst Time:** The total time required for the process to complete its execution.
- **Priority:** A value that determines the importance of the process (default is 0 if not specified). Used in the Priority Scheduling algorithm.
- **Time Quantum:** A fixed value Specifies a fixed execution time for each operation. Used exclusively in the Round Robin scheduling algorithm.

System Overview (Outputs)

After applying the selected scheduling algorithm, the system generates the following outputs for each process:

- **Response Time:** The time between the arrival of a process and the first time it starts execution.
- **Turnaround Time:** The total time taken for a process to complete.
- **Completion Time:** The time at which the process finishes its execution.
- **Waiting Time:** The total time a process spends waiting in the ready queue before execution.

In addition to individual process outputs, the system provides the following overall performance metrics:

- **Average Turnaround Time:** The average of all processes' turnaround times.
- **Average Waiting Time:** The average time all processes spend waiting in the ready queue.
- **Throughput:** The number of processes completed within a specific time period.

Tools Used

To develop the CPU Scheduling project, the following tools and technologies were utilized:

- **Java:** The core programming language used to implement the scheduling logic and handle process management.
- **ntelliJ IDEA:** Used as the primary development environment for writing and debugging Java code.
- **Scene Builder:** Used to design and build the graphical user interface (GUI) of the application, making it easier to create an interactive and visually appealing user experience.
- **JavaFX:** Used to create a modern and interactive user interface, allowing the display of process execution timelines and performance metrics.

Implementation

CPU Scheduling Algorithms

This project implements four key CPU scheduling algorithms, each with a unique approach to process management and execution:

1. **First Come, First Served (FCFS)**
   - **Concept:** Executes processes in the order they arrive.
   - **Key Point:** Simple but may cause delays if long processes arrive first (Convoy Effect).
2. **Shortest Job First (SJF)**
   - **Concept:** Executes the process with the shortest burst time first.
   - **Key Point:** Reduces waiting time but requires knowledge of burst times in advance.
3. **Round Robin (RR)**
   - **Concept:** Each process gets a fixed time slice (quantum) in a cyclic order.
   - **Key Point:** Fair for all processes, but performance depends on the time quantum value.
4. **Priority Scheduling**
   - **Concept:** Executes processes based on their assigned priority.
   - **Key Point:** Higher-priority processes run first, but lower-priority processes may face delays.

Implementation

1. First Come, First Served (FCFS)

- **Concept:** The simplest scheduling algorithm where processes are executed in the order they arrive (like a queue at a ticket counter).

- **How it works:**

1. Processes are sorted in a Queue based on their arrival time.
2. The process that arrives first is executed first, and the next process starts after the previous one finishes and so on.
3. No preemption (a running process cannot be stopped until it finishes).

- **Example:**

Process	Arrival Time	Burst Time	Execution Order
P1	0	4	0 → 4
P2	2	3	4 → 7
P3	3	2	7 → 9

Implementation

2. Shortest Job First (SJF)

- **Concept:** The process with the shortest burst time is executed first.

- **How it works:**

1. When the CPU becomes available, it selects the process with the shortest burst time.
2. If two processes have the same burst time, arrival time is used as a tiebreaker.

- **Example:**

Process	Arrival Time	Burst Time	Execution Order
P1	0	7	2 → 9
P2	2	4	0 → 4
P3	4	1	4 → 5

Implementation

3. Round Robin (RR)

- **Concept:** Each process is assigned a fixed time slice (quantum) and processes are executed in a cyclic manner.
- **How it works:**

1. All processes are placed in a circular queue.
2. Each process gets a fixed amount of CPU time (time quantum), after which it is preempted and moved to the back of the queue.
3. This continues until all processes are complete.

- **Time Quantum:** The maximum amount of time a process can run before being paused.

- **Example** (Time Quantum = 3):

Process	Burst Time	Execution Order (Time = 0)
P1	5	P1(3) → P2(3) → P3(3) → P1(2)
P2	3	Finished after 3 units
P3	8	P3(3) → P3(2)

Implementation

4. Priority Scheduling

- **Concept:** Processes are assigned a priority, and the process with the highest priority is executed first.
- **How it works:**

1. Each process is assigned a priority value (smaller number = higher priority).
2. The process with the highest priority runs first.
3. If two processes have the same priority, arrival time is used as a tiebreaker.

- **Example** (Priority values: Lower number = Higher Priority):

Process	Arrival Time	Burst Time	Priority	Execution Order
P1	0	7	2	2 → 9
P2	2	4	1	0 → 4
P3	4	1	3	4 → 5

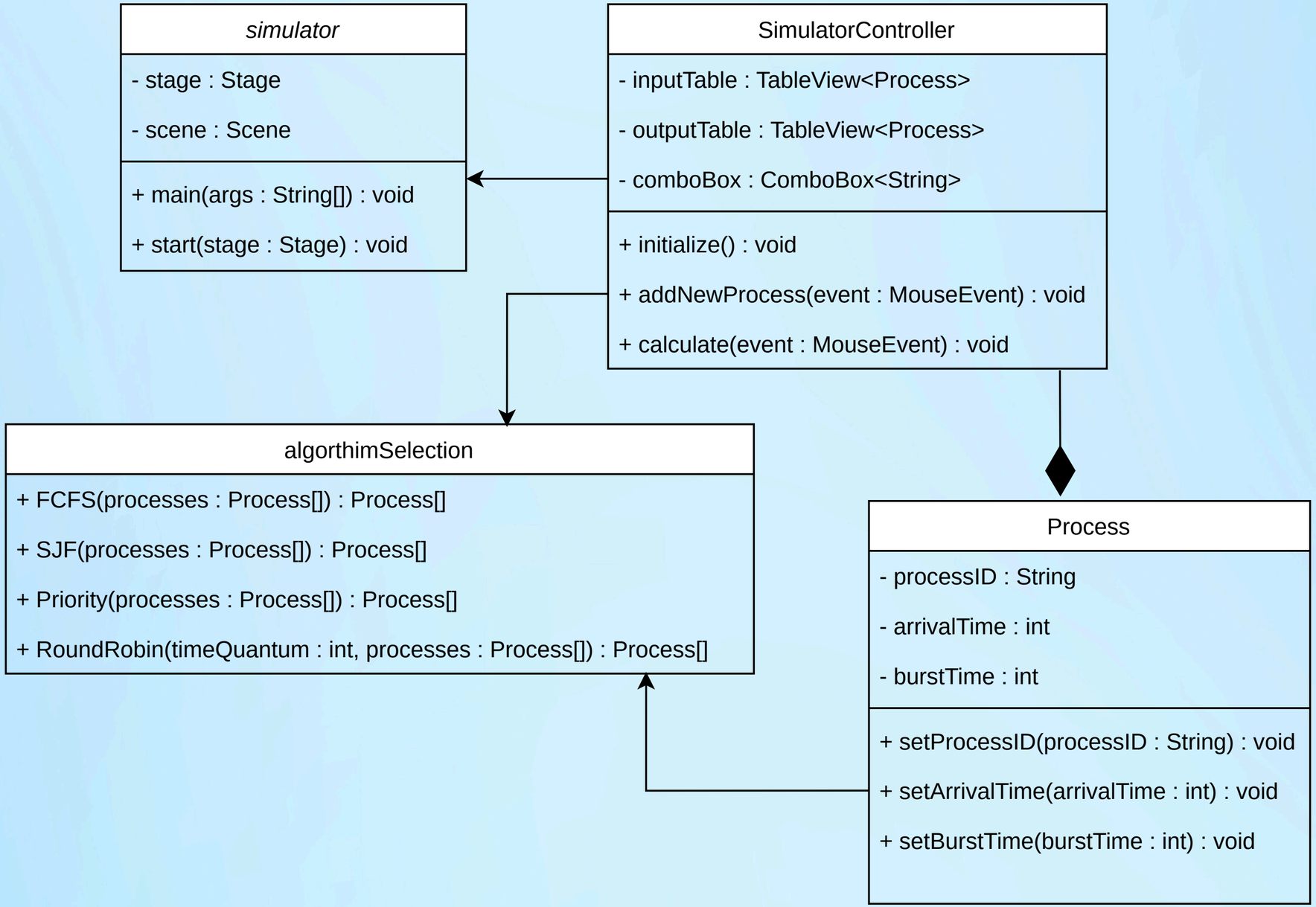
Implementation

Summary of Algorithm Differences

Algorithm	Key Factor	Performance
FCFS	Arrival Time	Simple but causes Convoy Effect
SJF	Shortest Burst Time	Best Turnaround Time
Round Robin	Time Quantum	Fair, but higher overhead
Priority	Priority Value	Higher-priority jobs run first

Functionality

UML Class Diagram



Functionality

Class 1 - Simulator

Class Name: Simulator

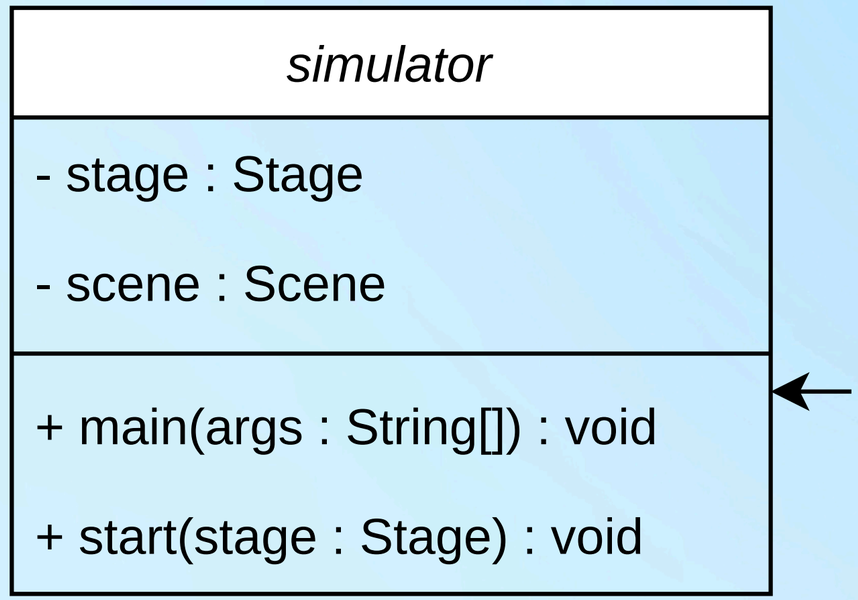
Purpose: The Simulator class serves as the entry point of the application. It is responsible for loading the graphical user interface (GUI) and launching the application

Key Methods:

- **main()**: Launches the application.
- **start(stage: Stage)**: Loads and displays the GUI window.

Key Responsibilities:

- Loads the FXML file (Simulator.fxml) that contains the user interface layout.
- Sets up the Stage and Scene with title, width, height, and styles.
- Starts the application using the main() method.



Functionality

Class 2 - SimulatorController

Class Name: SimulatorController

Purpose: The SimulatorController class controls the behavior of the user interface (UI). It allows users to add, edit, and delete processes. It also triggers scheduling algorithms and displays results.

Key Responsibilities:

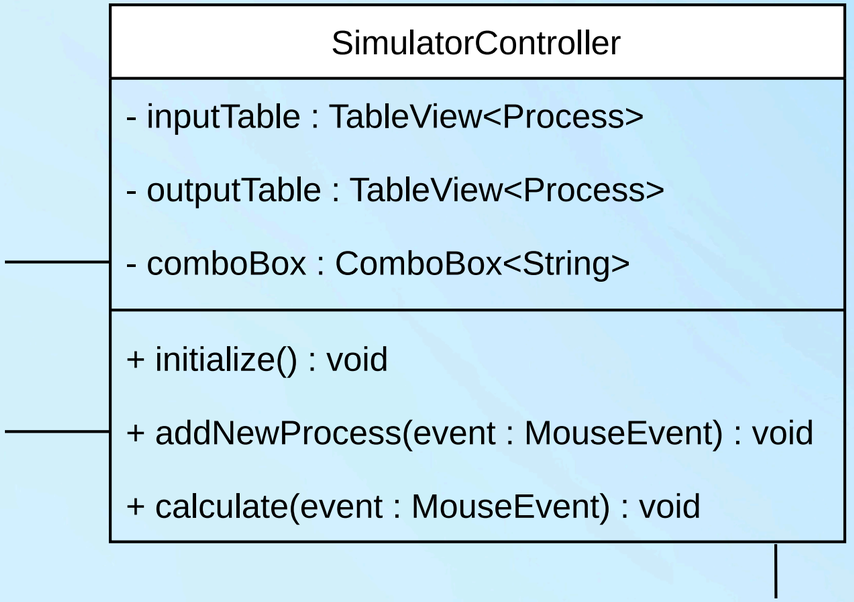
- Handles user input via buttons and text fields.
- Manages the input table (inputTable) and output table (outputTable).
- Calls scheduling algorithms from AlgorithmSelection class.

Key Attributes:

- **inputTable**: TableView to display user-added processes.
- **outputTable**: TableView to display the results of the scheduling.
- **comboBox**: Dropdown menu to select the scheduling algorithm.

Key Methods:

- **addNewProcess()**: Adds a new process to the input table.
- **calculate()**: Executes the selected scheduling algorithm.
- **clearAllProcess()**: Clears the input and output tables.



Functionality

Class 3 - Process

Class Name: Process

Purpose: The Process class represents each process in the CPU scheduling system. Each process has essential attributes like arrival time, burst time, priority, etc.

Key Attributes:

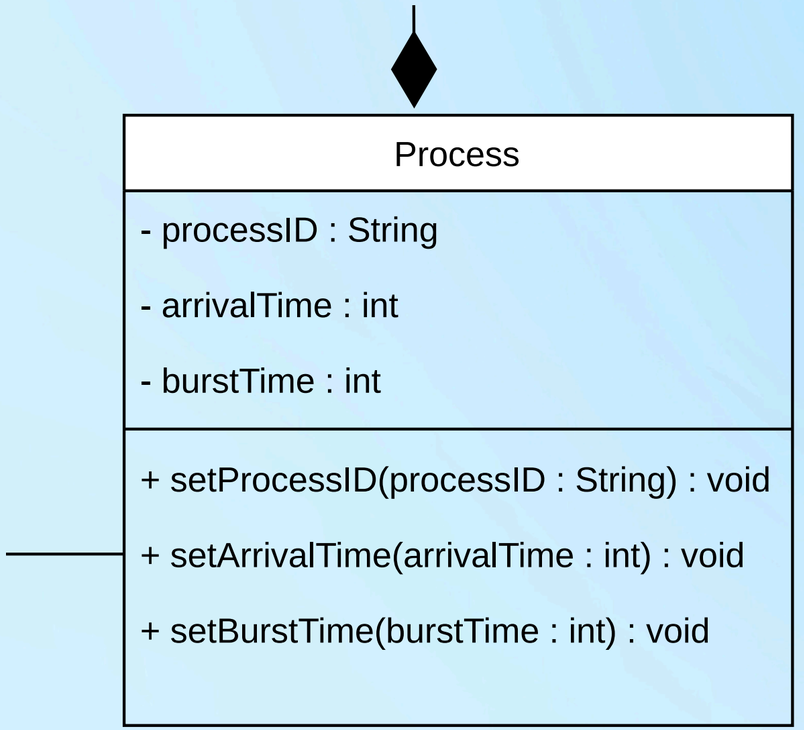
- **processID**: Unique identifier for each process.
- **arrivalTime**: Time when the process arrives in the system.
- **burstTime**: Total execution time required by the process.
- **priority**: Priority of the process (used in Priority Scheduling).

Key Methods:

- **setProcessID()**: Sets the process ID.
- **setArrivalTime()**: Sets the arrival time.
- **setBurstTime()**: Sets the burst time.

Key Responsibilities:

- Stores process data, such as processID, arrivalTime, and burstTime.
- Provides getters and setters to access and modify the process data.



Functionality

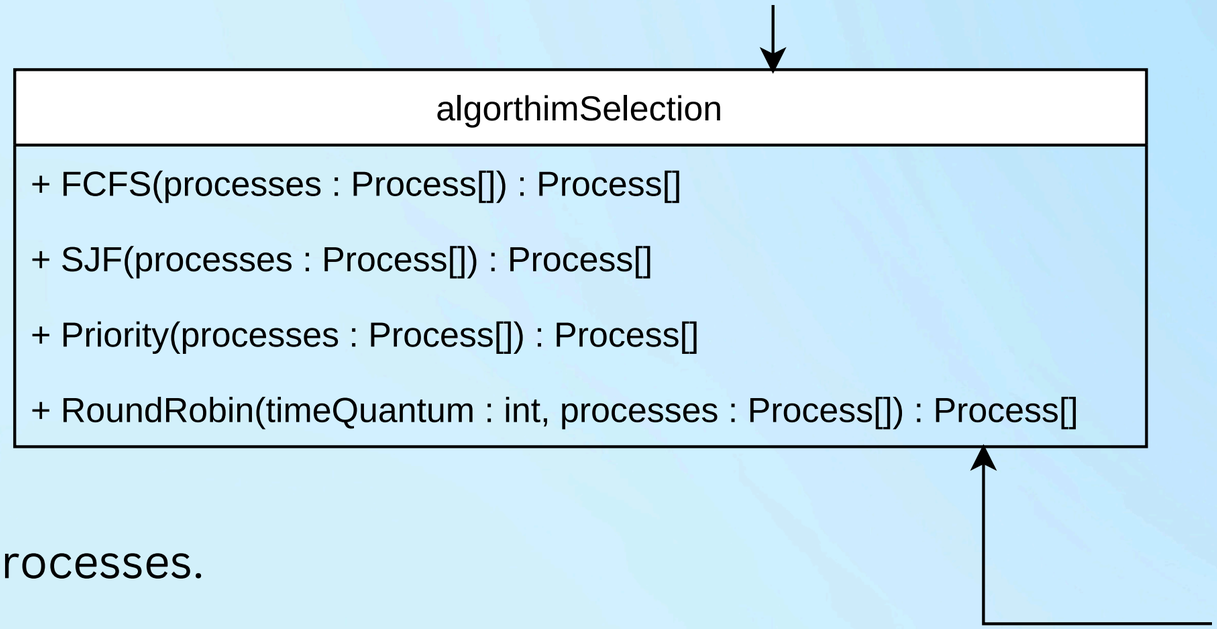
Class 4 - AlgorithmSelection

Class Name: AlgorithmSelection

Purpose: The AlgorithmSelection class contains the logic for scheduling algorithms. It implements the main CPU scheduling techniques: FCFS, SJF, Priority, and Round Robin

Key Responsibilities:

- Executes the scheduling algorithms.
- Calculates the Throughput, Turnaround Time, and Waiting Time for the processes.



Key Methods:

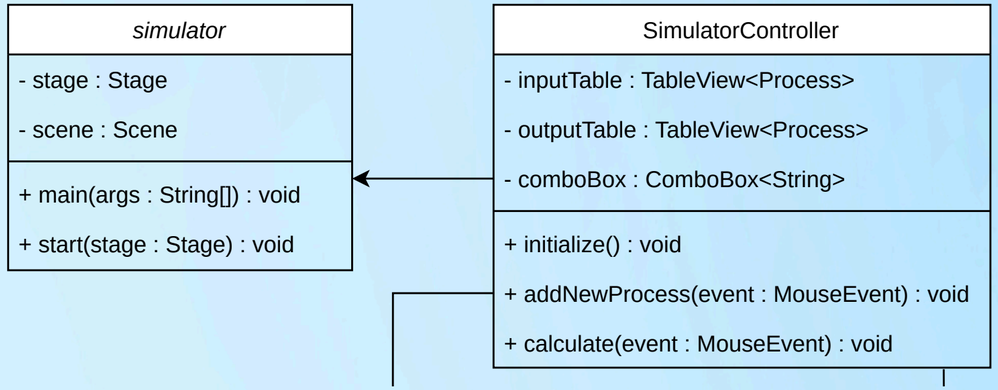
- **FCFS()**: First Come, First Serve scheduling.
- **SJF()**: Shortest Job First scheduling.
- **Priority()**: Priority-based scheduling.
- **RoundRobin()**: Round Robin scheduling.
- **calcThroughput()**: Calculates throughput for all processes.

Functionality

Relationships Between Classes

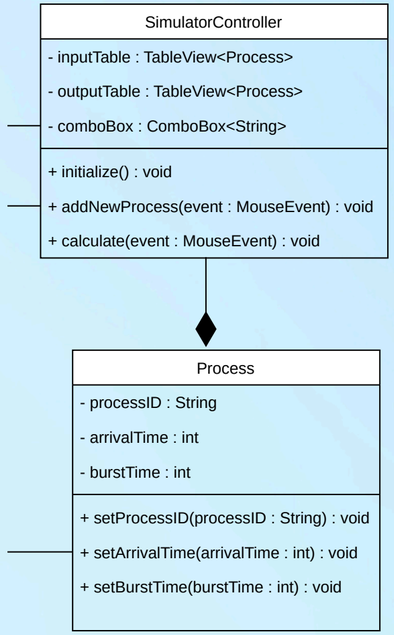
1. Dependency:

- It means that one class depends on another class to do its job.
- **Example:** Simulator → SimulatorController means that Simulator relies on SimulatorController to manage the user interface and handle user actions.



2. Aggregation :

- It means that one class "owns" or "has" objects of another class.
- **Example:** SimulatorController ◇→ Process means that the controller "has" multiple process objects stored inside the inputTable and outputTable.



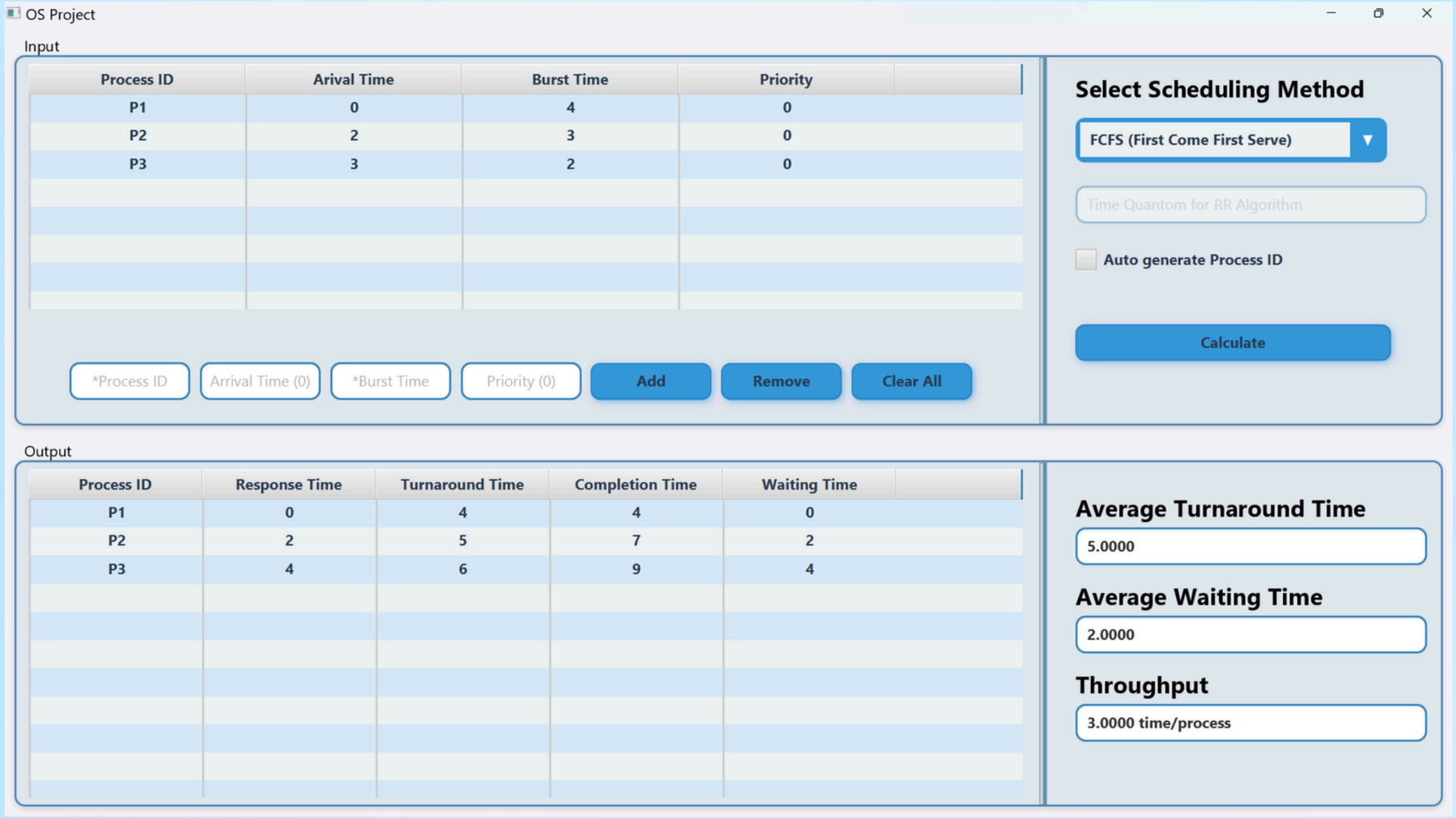
Functionality

Relationships Between Classes

Relationship	Type	Meaning	Explanation
Simulator → SimulatorController	Dependency	Uses	Simulator depends on SimulatorController to control the UI.
SimulatorController ◇→ Process	Aggregation	Has-a	SimulatorController "has" a list of processes in the input/output tables.
SimulatorController → AlgorithmSelection	Dependency	Uses	SimulatorController calls AlgorithmSelection to run scheduling algorithms.
AlgorithmSelection → Process	Dependency	Uses	AlgorithmSelection applies scheduling logic to Process objects.

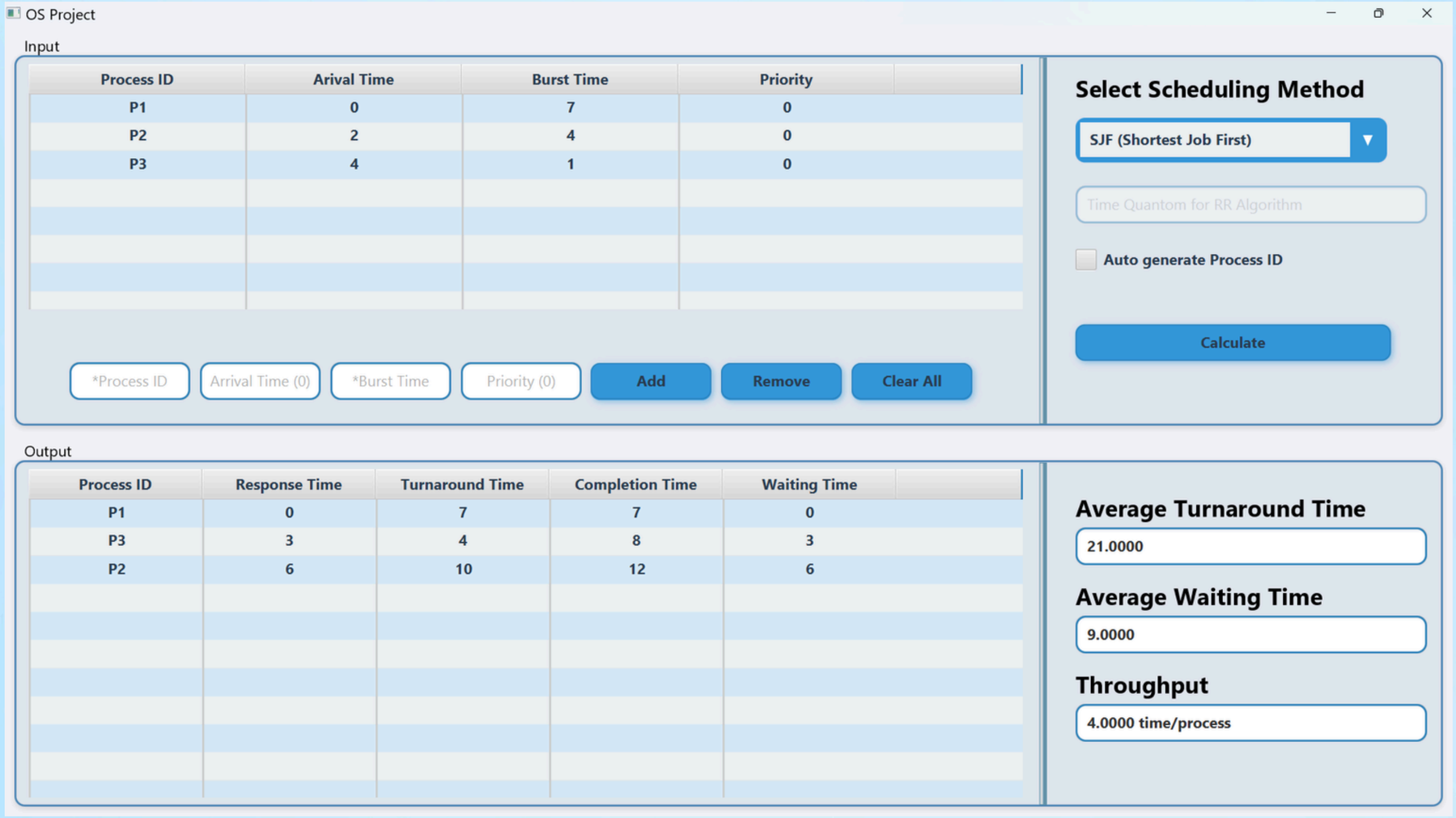
Screenshots

Implementing FCFS Algorithm



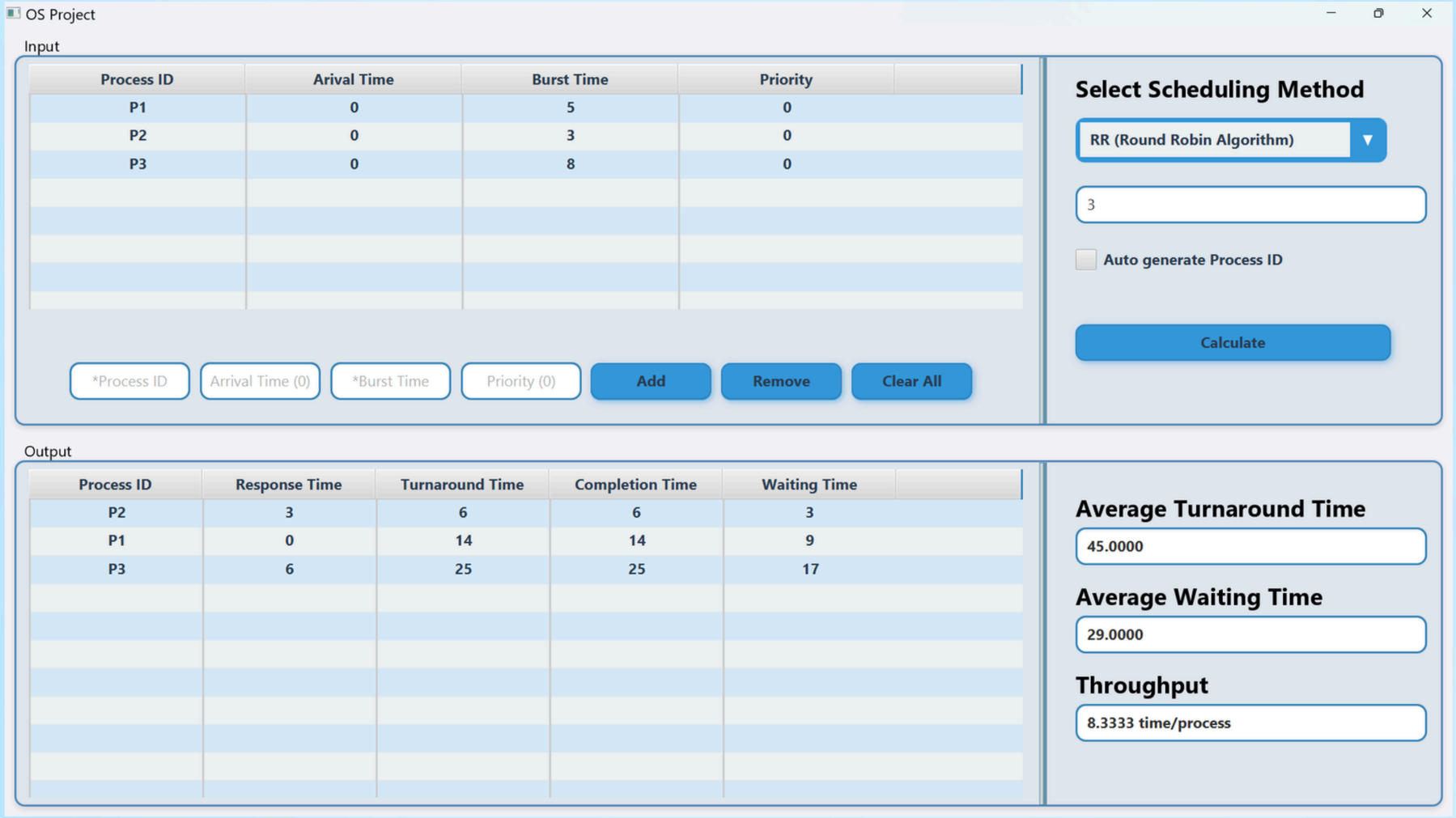
Screenshots

Implementing SJF Algorithm



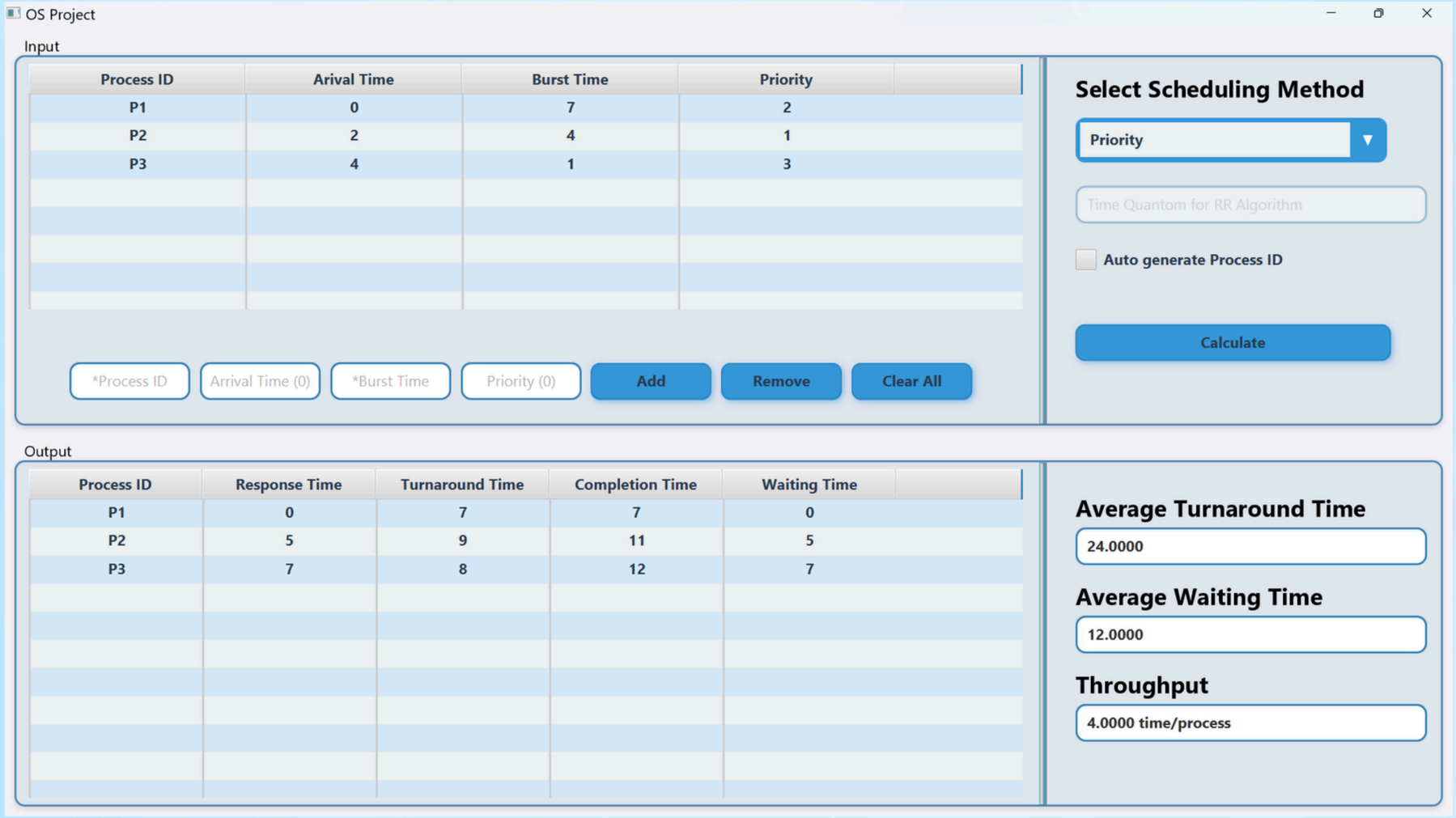
Screenshots

Implementing RR Algorithm



Screenshots

Implementing Priority Algorithm



Conclusion

This project focused on the design and implementation of a **CPU Scheduling System** that simulates the execution of key scheduling algorithms, including **FCFS, SJF, Round Robin, and Priority Scheduling.** Each algorithm was explained in detail, highlighting its logic, execution flow, and impact on system performance.

We also provided insights into the development process, showcasing the tools and technologies used, such as **IntelliJ IDEA, Scene Builder, Java, and JavaFX.** The project's implementation demonstrates how different scheduling approaches influence metrics like **waiting time, turnaround time, and throughput.**

By the end of this presentation, we have gained a clear understanding of CPU scheduling techniques and the practical steps involved in developing a scheduling system. This knowledge is essential for building efficient, fair, and responsive operating systems.

By: Ahmed Yousef Mohamed 2201789

CPU Scheduling project Documentation

Operating System Subject

Video