

# Processor Execution Simulator

# ATYPON

**Author:** Ahmad Emad

**GitHub:** [Processor-Execution-Simulator](#)

**YouTube Video:** [Demonstartion](#)

**Email:** [ahmademad995.ae@gmail.com](mailto:ahmademad995.ae@gmail.com)

## Objective Overview:

The Processor Execution Simulator is designed to simulate the execution of tasks across multiple synchronized processors. It ensures efficient task scheduling based on priority and execution time while adhering to real-time task creation. The project demonstrates multithreading, priority-based scheduling, and concurrent data management. Finally outputs a detailed cycle-by-cycle execution report.

# Table of Contents

<b>Introduction</b>	3
Overview	3
<b>Object-Oriented Design</b>	3
UML-Diagram	3
<b>Specifications</b>	4
System Overview	4
Input Parameters	4
Structure of input file for tasks information:	5
Processor Behavior	5
Scheduling Policy	5
Simulation Execution	5
Output Format	6
<b>Methodology</b>	6
Implementation Details	6
Data Structures Used	6
Key Functions	7

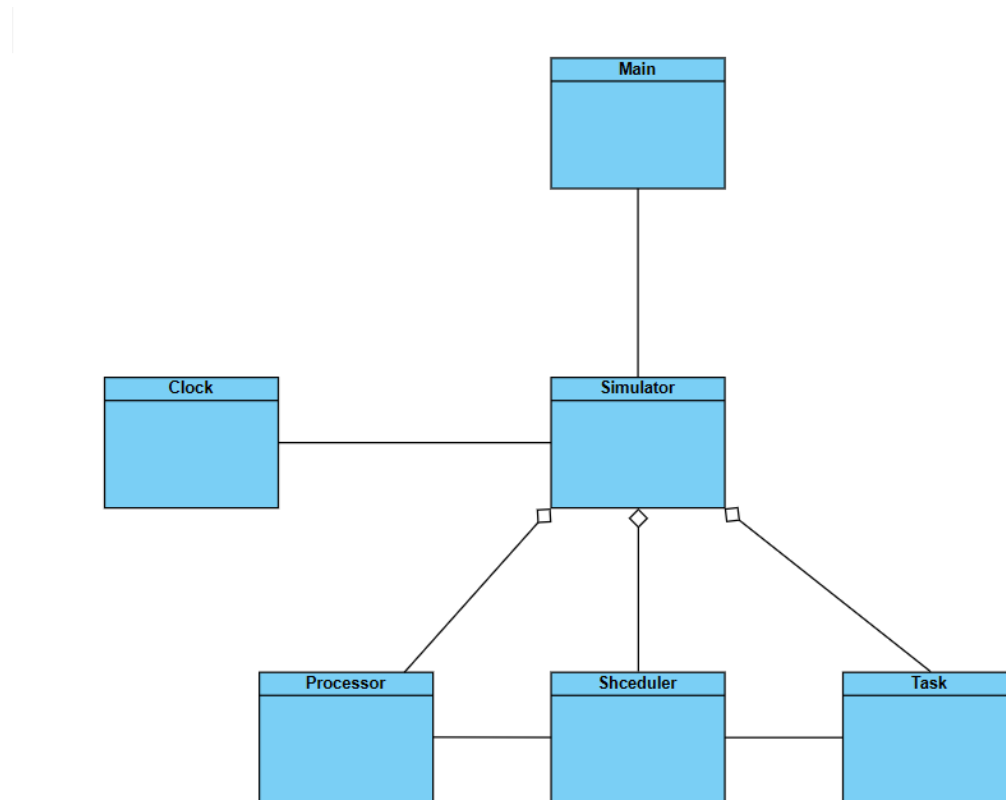
# Introduction

## Overview

The Processor Execution Simulator is a multithreaded system designed to model the execution of tasks across multiple synchronized processors. The simulator processes a set of tasks, each defined by its creation time, execution time, and priority. It manages task scheduling using a priority-based queue and assigns tasks to available processors. The simulation runs for a fixed number of clock cycles, generating a cycle-by-cycle report that tracks task creation, execution, completion and processors' state. This report provides insights into processor utilization and scheduling efficiency, demonstrating key concepts in concurrent programming and real-time task management.

## Object-Oriented Design

### UML-Diagram



## Design Features

- **SOLID Principles:** The design adhere to the SOLID principles to ensure a robust and maintainable system architecture.
- **Modular Design:** Classes such as Task, Processor, Clock, and Scheduler each handle a specific responsibility.
- **Singleton Design Pattern:** The Singleton pattern is applied where a single instance of a class is required, ensuring controlled access to shared resources like the Clock, Simulator, and Scheduler
- **Flexibility:** The design allows easy modification and extension of the simulation's functionality.
- **Extensibility:** The system support the addition of new features, such as different scheduling algorithms or additional task properties, without requiring major changes to the existing codebase.
- **Maintainability:** The is organized and written in a way that makes it easy to understand, debug, and update. This includes adhering to best practices in object-oriented design and programming.

## Specifications

### System Overview

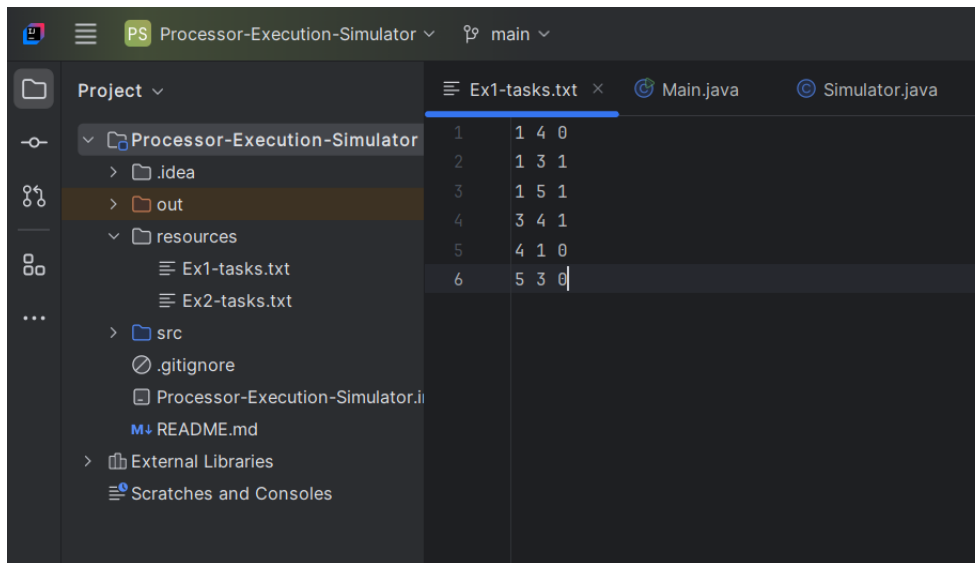
- The simulator models multi-processor execution for handling tasks based on priority and execution time.
- Processors operate in synchronized clock cycles, ensuring consistent scheduling.
- Tasks are stored in a priority queue before being assigned to available processors.

### Input Parameters

The simulator requires three command-line arguments:

1. **Number of processors (int):** Defines the fixed number of processors.
2. **Total number of clock cycles (int):** Specifies the duration of the simulation.
3. **Input file path for tasks information (string):** Path to a text file containing task information.

## Structure of input file for tasks information:



Each task in the input file is represented by three space-separated integers:

<creation\_time> <execution\_time> <priority>

1. Creation Time (int) – The cycle in which the task is created.
2. Execution Time (int) – The number of cycles required for task completion.
3. Priority (int) – 1 for high priority, 0 for low priority.

## Processor Behavior

- Each processor can execute only one task at a time.
- Task execution is non-preemptive (once assigned, it runs to completion).

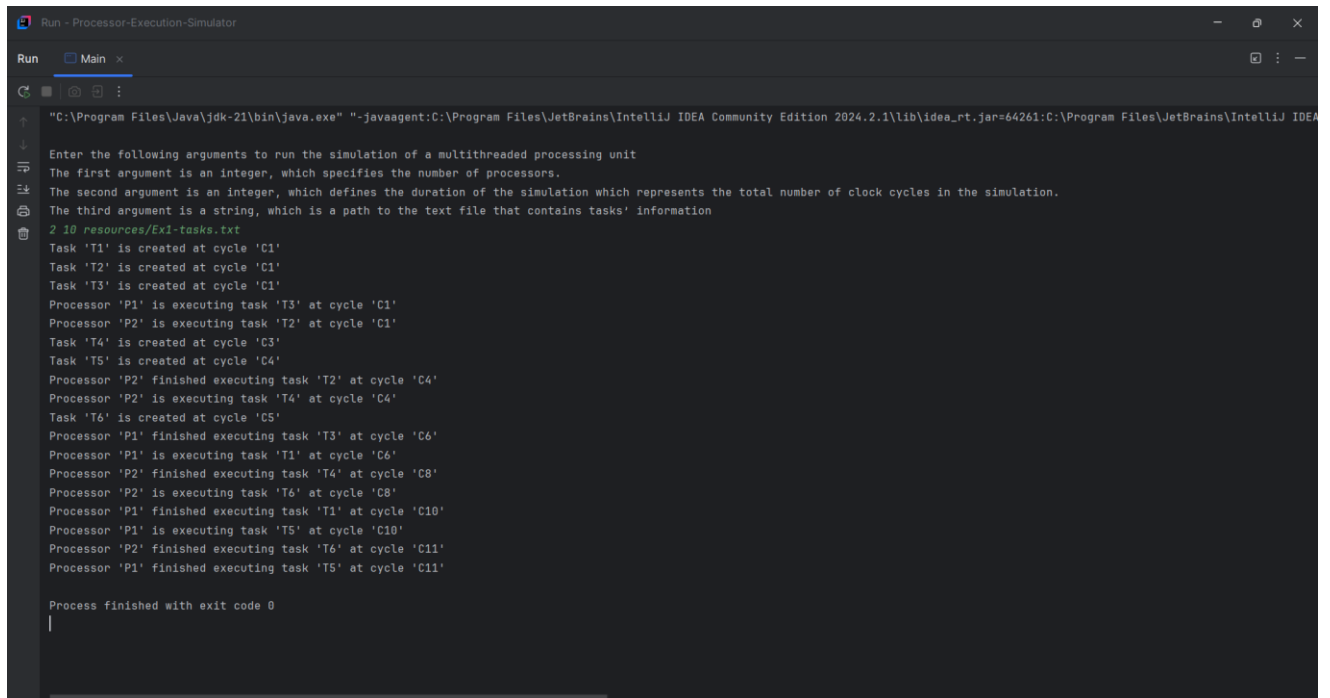
## Scheduling Policy

- High-priority tasks are scheduled before low-priority tasks.
- Among tasks with the same priority:
  - Tasks with longer execution times are scheduled first.
  - If execution times are equal, a task is chosen randomly.
- Idle processors immediately pick the highest-priority task from the queue.

## Simulation Execution

- The simulator runs for the specified number of clock cycles.
- At each cycle, it updates task states, assigns available processors, and generates a real-time console report.

## Output Format



```
Run - Processor-Execution-Simulator
Main x
C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.1\lib\idea_rt.jar=64261:C:\Program Files\JetBrains\IntelliJ IDEA
Enter the following arguments to run the simulation of a multithreaded processing unit
The first argument is an integer, which specifies the number of processors.
The second argument is an integer, which defines the duration of the simulation which represents the total number of clock cycles in the simulation.
The third argument is a string, which is a path to the text file that contains tasks' information
2 10 resources/Ex1-tasks.txt
Task 'T1' is created at cycle 'C1'
Task 'T2' is created at cycle 'C1'
Task 'T3' is created at cycle 'C1'
Processor 'P1' is executing task 'T3' at cycle 'C1'
Processor 'P2' is executing task 'T2' at cycle 'C1'
Task 'T4' is created at cycle 'C3'
Task 'T5' is created at cycle 'C4'
Processor 'P2' finished executing task 'T2' at cycle 'C4'
Processor 'P2' is executing task 'T4' at cycle 'C4'
Task 'T6' is created at cycle 'C5'
Processor 'P1' finished executing task 'T3' at cycle 'C6'
Processor 'P1' is executing task 'T1' at cycle 'C6'
Processor 'P2' finished executing task 'T4' at cycle 'C8'
Processor 'P2' is executing task 'T6' at cycle 'C8'
Processor 'P1' finished executing task 'T1' at cycle 'C10'
Processor 'P1' is executing task 'T5' at cycle 'C10'
Processor 'P2' finished executing task 'T6' at cycle 'C11'
Processor 'P1' finished executing task 'T5' at cycle 'C11'
Process finished with exit code 0
```

The simulator logs key events per clock cycle, including:

- Task Creation: When a task enters the queue.
- Task Assignment: When a processor starts executing a task.
- Task Completion: When a task finishes execution.
- Processor State Updates: Showing processors that are idle.

## Methodology

### Implementation Details

- Utilizes multithreading for concurrent task scheduling and execution.
- Ensures thread safety with proper synchronization mechanisms.
- Implements an efficient priority queue to optimize task scheduling.

### Data Structures Used

- **Priority Queue:** Manages task scheduling based on priority and execution time.
- **Synchronized Clock:** Ensures all processors execute in sync.
- **Processor Pool:** Tracks available and busy processors.

## Key Functions

- **assignTasks():** Reads task data from the file and inserts it into the queue once creation time is reached.
- **assign():** Assigns task to processor and updates processor state.
- **executing:** Runs tasks until completion synchronizing with the clock.
- **compareTo():** implements Comparable<> in class Task to prioritize high priority tasks and available processors in class Processor.