Processor Execution Simulator Project Report

Mohannad Al-Tamimi

mohanad.tamimi81@gmail.com

3rd Project

Introduction

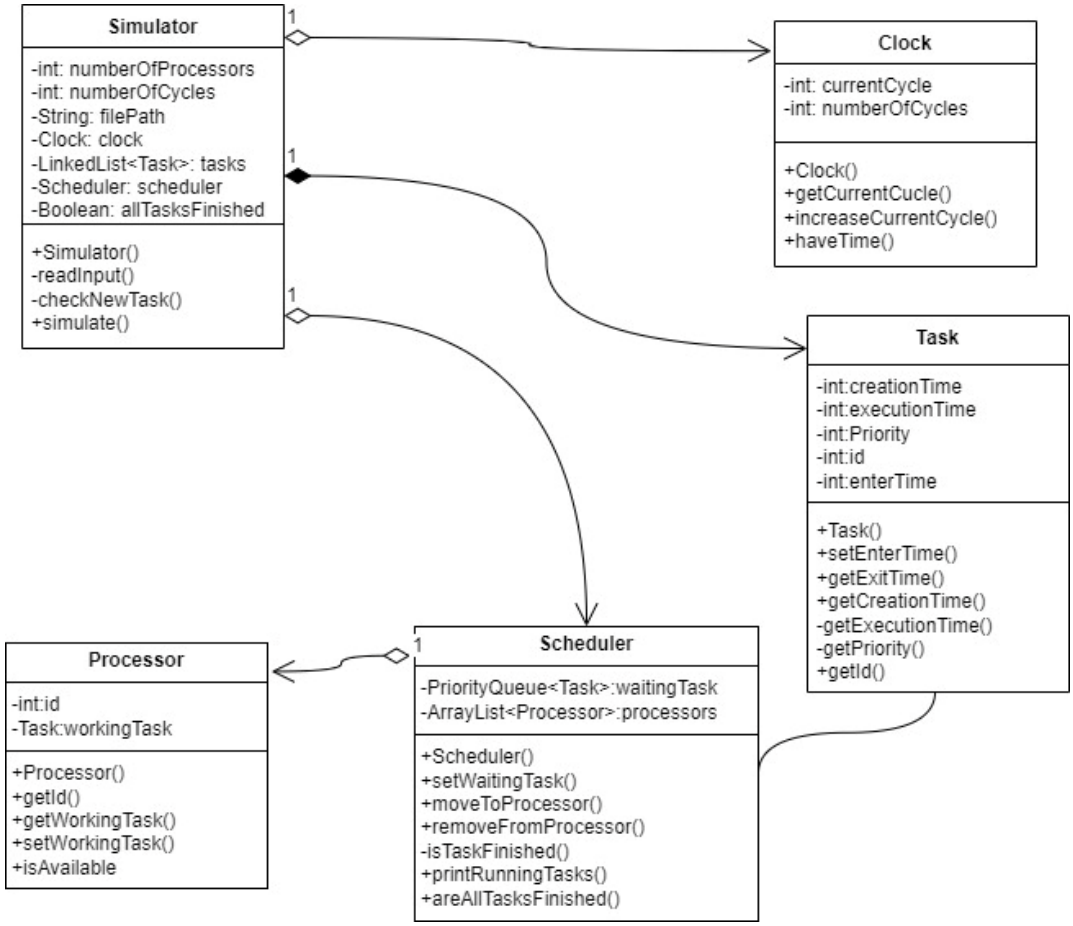
In this assignment we are required to build a simulator that simulates how a processor handles task execution. In this report I will introduce my implementation.

Implementation

We were required to have at least these classes which were:

Main, Clock, Processor, Simulator, and Task

I first want to start by presenting the UML diagram that represents these classes and how they interact with each other



The project consists of five main classes: `Clock`, `Processor`, `Scheduler`, `Task`, and `Simulator`. These classes interact to create a process simulator that models the execution of tasks on multiple processors over a series of cycles. The simulation involves reading input from the user, managing tasks, and producing cycle-by-cycle reports on task execution.

The simulation process follows these steps:

1. Task Arrival Check: Check if a new task arrives at the current cycle and transfer it to the scheduler if true.

2. Task Scheduling: Check if the scheduler has waiting tasks and if any processor is available. Transfer tasks to processors if both conditions are met.

3. Task Completion Check: Check if any task finishes its execution in a processor. Remove the completed task from the processor if true.

This process repeats as long as there are tasks waiting to be executed.

Design Principles

The project employs object-oriented design principles to ensure readability, maintainability, and reusability.

1. Abstraction:

The project abstracts the functionality of each class, allowing interactions at a higher level without exposing implementation details. For example, the `Simulator` class calls the `simulate` method, which abstracts the entire simulation process without revealing the internal workings.

2. Encapsulation

The project ensures encapsulation by making class data members private and providing methods as interfaces to interact with these members. This approach secures the data and maintains the code. For instance, the `Clock` class has private data members and provides public methods to manipulate these members appropriately.

Algorithms and Data Structures

1. LinkedList

The `tasks` list is implemented using a `LinkedList` to efficiently handle additions and deletions.

```
private final LinkedList<Task> tasks;
```

2. ArrayList

The `processors` list uses an `ArrayList` to facilitate easy traversal.

```
private final ArrayList<Processor> processors;
```

3. PriorityQueue

The `waitingTask` queue is implemented using a `PriorityQueue` to automatically sort tasks based on their priority and other conditions.

```
private final PriorityQueue<Task> waitingTask;
```

Algorithms

- ### Task Scheduling Algorithm

The `Scheduler` class uses a priority queue to manage tasks, prioritizing tasks based on their priority and execution time. The `compareTo` method in the `Task` class ensures tasks are sorted correctly.

```
    @Override
     public int compareTo(Task o){
        if (getPriority()>o.getPriority()){
            return -1;
        }
        if(getPriority()<o.getPriority()){
            return 1;
        } else {
            return Integer.compare(o.getExecutionTime(),getExecutionTime());
        }
    }
}
```

- ## Processor Allocation Algorithm

The `Scheduler` checks if processors are available and assigns tasks from the priority queue to available processors. This ensures efficient task execution and maximizes processor utilization.

- ## Task Completion Check

The `Scheduler` also checks if tasks have completed their execution in each cycle and removes them from processors, ensuring tasks do not occupy processors longer than necessary.

- ## Simulation Control Algorithm

The `Simulator` controls the overall flow of the simulation, advancing the clock and triggering task scheduling and completion checks. This ensures a smooth and orderly simulation process.

In conclusion, I successfully used object-oriented design principles with the needed data structures to implement a processor execution simulator that is readable and scalable. We also used the console to report what is happening inside the processors in each cycle from task creation and execution, to finishing said tasks and removing them from the processor to move a new task in its place.