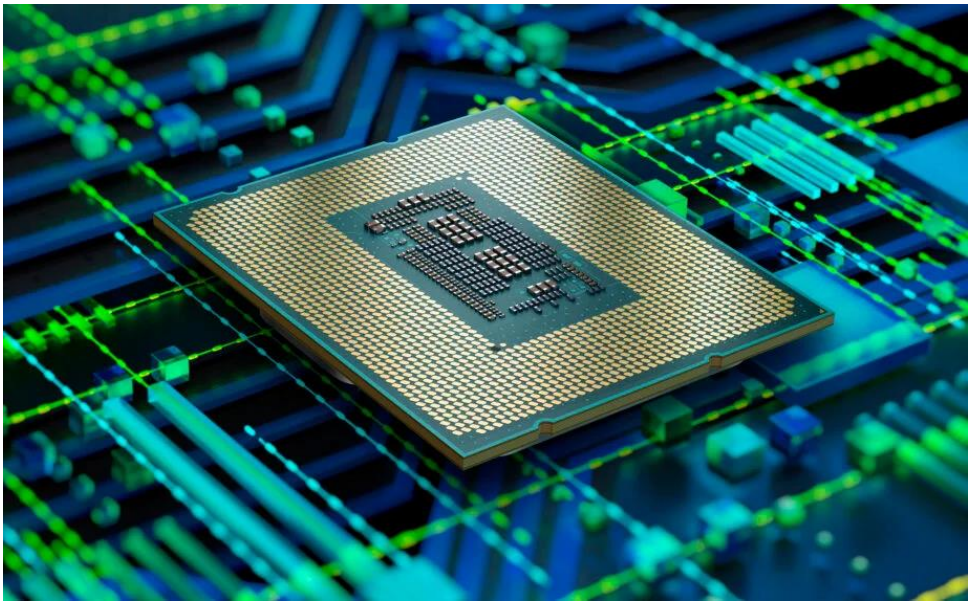


PROCESSOR EXECUTION SIMULATION

Bahaeddin Ahmad Ibrahim Hammad
Atypon

Introduction

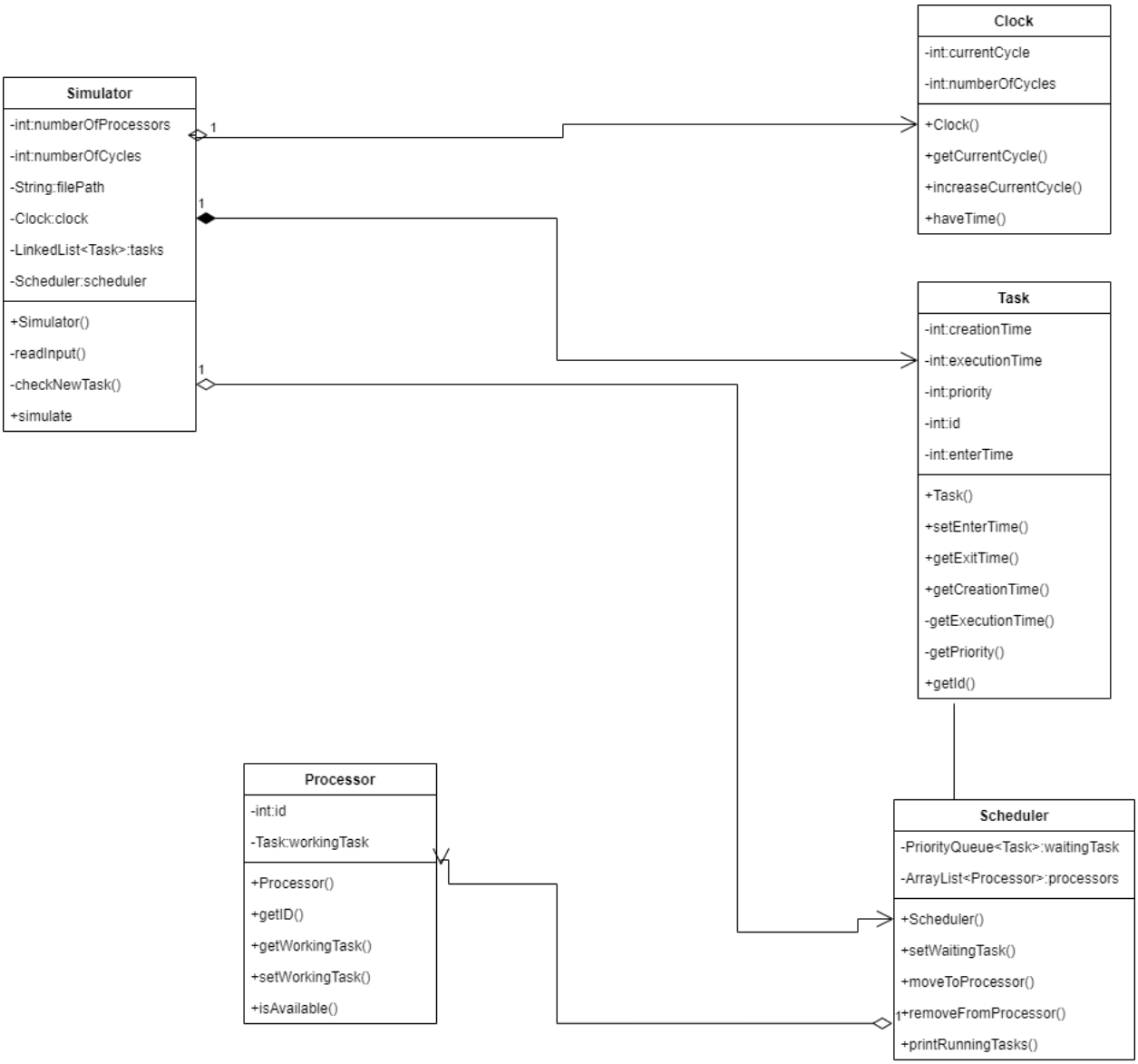
In this assignment, we are required to build a simulator that simulates processor execution for processes. This report will discuss my own implementation of the assignment.

Implementation

This project simulates the work of the CPU, so it contains these classes:

- Main
- Clock
- Processor
- Simulator
- Task

The UML diagram for the project:



Review the code from a high-level view.

First, we have 5 main classes in our project (Clock, Processor, scheduler, Task, Simulator), and the interactions between these classes will create the process simulator, so first we read the input from the user after that we start simulating the process in the following way:

1. will check if a new task enters at this time, if that's true will transfer the task to the scheduler.

2. then, we will check if the scheduler has a waiting task and if any processor is empty, if that's true will transfer the task to the processor.

3. Then, we will check if any task ends its work in the processor, and if that's true will remove the task from the processor.

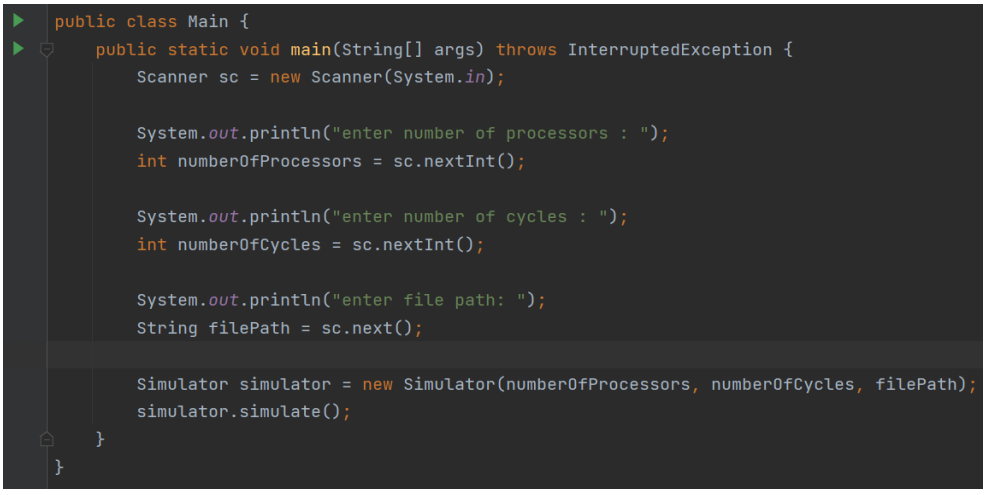
We will repeat the process as long as there are some tasks waiting to be executed

the design used in this project.

We used object-oriented programming design principles in this project as:

1. Abstraction: we used abstraction in such a way that dealing with the objects is done at a higher-level view so each object has an interface that we can use to interact with the object, so by using abstraction we make our code more readable, maintainable and reusable

As an example



```
public class Main {
    public static void main(String[] args) throws InterruptedException {
        Scanner sc = new Scanner(System.in);

        System.out.println("enter number of processors : ");
        int numberOfProcessors = sc.nextInt();

        System.out.println("enter number of cycles : ");
        int numberOfCycles = sc.nextInt();

        System.out.println("enter file path: ");
        String filePath = sc.next();

        Simulator simulator = new Simulator(numberOfProcessors, numberOfCycles, filePath);
        simulator.simulate();
    }
}
```

As we can see, a simulator object is called a simulation method, knowing that it will simulate our project without knowing how it will simulate.

2. Encapsulation: we used encapsulation in such a way that we ensure abstraction by making the data to be private in the classes and providing methods as an interface to deal with the data associated with the objects, by doing that we ensure that the data will be secure, and our code is maintainable.

As example

```
public class Clock {
    4 usages
    private int currentCycle;
    2 usages
    private final int numberOfCycles;

    1 usage
    Clock(int numberOfCycles) {
        currentCycle = 1;
        this.numberOfCycles = numberOfCycles;
    }

    3 usages
    public int getCurrentCycle() { return currentCycle; }

    1 usage
    public void increaseCurrentCycle() {
        currentCycle++;
    }

    1 usage
    public Boolean haveTime() {
        return currentCycle != (numberOfCycles + 1);
    }
```

As we can see the Clock class has two data members that are declared private so these data members can't be used outside the class, so we created a method as an interface that deals with data members in the appropriate way, and we apply these concepts to all classes to ensure abstraction and encapsulation.

Algorithms and Data Structures

1. I used the linked list data structure to store the tasks because deleting and adding will consume less time complexity.

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

2. I used ArrayList data structure to store the processors because traversing the processors is done easily using ArrayList.

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

3. I used a priority queue to store the waiting tasks because priorityQueue will sort the tasks depending on the conditions and the priorities of tasks and we will discuss this later in the algorithm section.

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

4. We used a String data structure to store the file path that we will read input from and for the rest of the attributes we used an integer primary data type as is the best data type for them.

Algorithms

- In the most of methods we just used a normal for/while loop to achieve the goal of the method

- We used the following algorithm in the compareTo method that we override from the comparable InterFace:

```
    @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());
        }
    }
}
```

Because the priority queue sorts the elements in increasing order so that the smallest one is in the front and the largest one is in the last we used the following algorithm, so we first compare the priority if the functions return -1 so which means the task will be smaller than the argument task so it will take a higher place and if their priority is equal we will look at the execution time.

THAT'S IT.

THANK YOU.