CS 321 Data Structures Project #2 (100 points)
CPU Scheduling Simulation using a Priority Queue

Introduction

In this project we will simulate the priority-based round robin scheduling algorithm for assigning a CPU to a running process. All operating systems run some version of this algorithm in their kernel (the core part of the operating system). The round robin scheduling algorithm works as follows.

1. A system defines a unit time (time slice).
2. A priority queue holds the processes waiting for CPU time.
3. Each process has a priority level, time remaining to finish, arrival time, and waiting time.
4. The system assigns the next time slice on the CPU to the highest priority process (if there is a tie, the one arriving first is picked) in the priority queue.
5. Each time a process gets a time slice on the CPU, its remaining time to finish should be decremented by one. For all other processes, we increment the waiting time by one.
6. In order to avoid starvation problem, the system will increment the priority for lower priority processes. In this simulation, if any process does not get any time slice for a certain amount of time (an input parameter), its priority will be incremented by one until it reaches the highest allowable priority level. Each time the priority is incremented, we will reset the waiting time back to zero.

In this project, we will implement a `MaxHeap` class using an array. Then we will implement a priority queue `PriorityQueue` by extending the `MaxHeap` class. Each node in the max-heap contains an instance of the `Process` class. The `Process` class implements `Comparable` interface so that the comparison between nodes in max-heap can be made by calling `compareTo` method. To implement the `compareTo` method, process with larger priority level has a higher priority. In case of a tie, process with earlier arrival time should be picked.
See Figure 1 on page 4 for the details of the scheduling algorithm.

Setup

Navigate to the class Git repository that you cloned for the last project and update it as follows:

```
cd CS321-resources
git pull
```

The project starter files are in the subfolder `projects/p2`. The main class is `CPUScheduling.java`. This file and the `Averager.java` should not be modified (for the test script to run properly)!

Description and Design

The following two classes are provided to be used as is without any changes.

1. `CPUScheduling.java:` simulates the round robin CPU scheduling algorithm. This the main class is already complete.
2. `Averager.java:` keeps track of the number of processes in the simulation and computes the average turn around time. This class is finished for us.

In addition to these two java classes, we should implement other classes for the CPU scheduling simulation. Several interfaces have been provided to make it easier to design the classes. Suggested other classes include, but are not limited to the following:

1. `Process.java` defines a process. Each process has a priority level, time remaining to finish, arrival time, and waiting time. The `Process` class should implement the `ProcessInterface` and the `Comparable` interface.
2. `ProcessGenerator.java` randomly generates processes with a given probability. At beginning of each time unit, whether a process will arrive is decided by the given probability. In addition, while generating a new process, both its priority and the required time units to finish the process are randomly generated within given ranges. The `Process` class should implement the `ProcessInterface`. It should provide two overloaded constructors, so that we can pass in the *probability* and, optionally, the *random seed.*
3. `MaxHeap.java` defines a max-heap. Each node in the max-heap contains a process. the pseudo-code in the textbook will have to be adapted to work with `Process` objects containing keys instead of just keys.
4. `MyPriorityQueue.java` extends the MaxHeap class to provide a priority queue and implements the `PriorityQueueInterface`.

What we need to do?

1. Create a new project in our favorite IDE.
2. Copy all the files from the files from the class Git repo `CS321-resources/projects/p2` into our project.
3. Write those suggested classes and additional classes if any.
4. To run the simulation, the following command-line arguments need to be provided:

   ```
   java CPUScheduling <maxProcessTime> <maxPriorityLevel> <timeToIncrementPriority>
   <simulationTime> <processArrivalRate> [<seed>]
   ```

   where

   - **maxProcessTime:** largest possible time units required to finish a process. That is, any process arrived will require at least 1 time unit and at most `maxPriorityLevel` time units to finish.

- **maxPriorityLevel:** highest possible priority in this simulation. That is, a process can have a priority, ranging from $1, 2, \ldots,$ maxPriorityLevel.
- **timeToIncrementPriority:** if a process didn't get any CPU time for this timeToIncrementPriority time units, the process's priority will be increased by one.
- **simulationTime:** the total time units for the simulation.
- **processArrivalRate:** A probability between 0 and 1. Use this rate to decide whether to generate a new process in each time unit.
- **seed:** A seed for random number generator. Useful for testing as providing the same seed will lead to the exact same simulation.

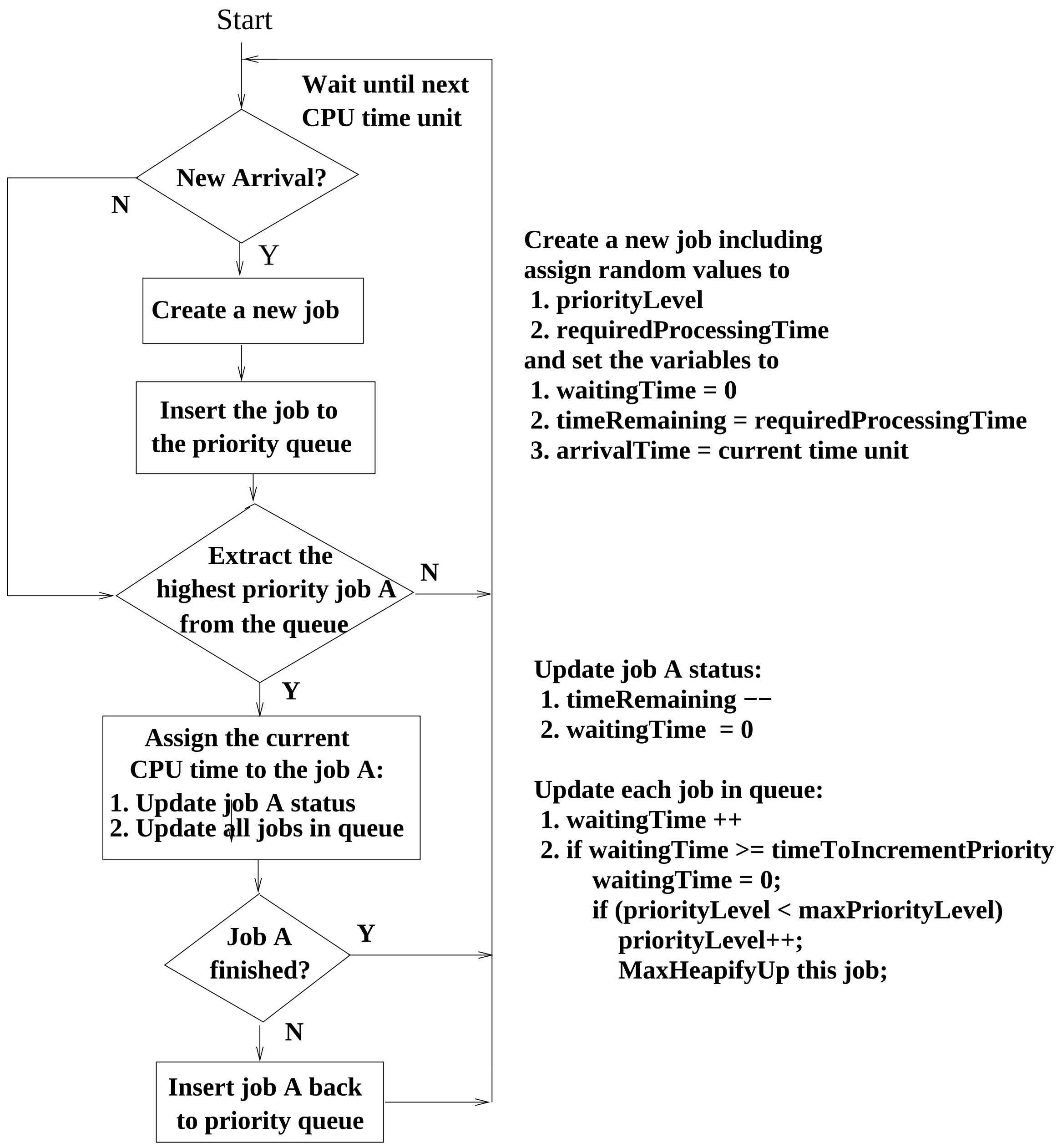
The simulation flow chart for each CPU time unit is given in the figure on the last page.

5. Sample input and output files can be found in the `test-cases/` subfolder of the project. That includes a test script named `run-tests.sh`.
6. Finish the `README.md` file. A template for the `README.md` file is provided in the class repository in the file `README_TEMPLATE.md` found in the `projects` folder in the class git repository.

Submission

Copy the project folder to `onyx`. Then remove all `*.class` files and other extra files. It should only have the java source files, the `run-tests.sh` script and test-cases subfolder, and the `README.md`. `Then submit as shown below FROM WITHIN the project directory on onyx using the following command.`

```
      submit amit CS321 p2
```



Start
Wait until next CPU time unit
New Arrival?
N
Y
Create a new job
Insert the job to the priority queue
Extract the highest priority job A from the queue
N
Y
Assign the current CPU time to the job A:
1. Update job A status
2. Update all jobs in queue
Job A finished?
Y
N
Insert job A back to priority queue
Create a new job including assign random values to
1. priorityLevel
2. requiredProcessingTime
and set the variables to
1. waitingTime = 0
2. timeRemaining = requiredProcessingTime
3. arrivalTime = current time unit
Update job A status:
1. timeRemaining −−
2. waitingTime = 0
Update each job in queue:
1. waitingTime ++
2. if waitingTime >= timeToIncrementPriority
waitingTime = 0;
if (priorityLevel < maxPriorityLevel)
priorityLevel++;
MaxHeapifyUp this job;

Figure 1: The Scheduling Algorithm