CS526 O2
Project Assignment

This project is an implementation of a small simulation program. It is a simulation of a process scheduler of a computer system. This simulated scheduler is a very small, simplified version, which reflects some of the basic operations of a typical process scheduler.

To successfully finish the program, students are expected to:

- ***Study*** and learn how to define and use a user-defined comparator.
- ***Study*** and learn how to use Java's *PriorityQueue*.
- ***Study*** and learn how to write a small simulation program.

Name the file with the main method *ProcessScheduling.java*. You may define other classes as needed in separate files.

The following describes the scheduling system that is simulated:

> Processes arrive at a computer system and the computer system executes the processes one at a time based on a priority criterion. Each process has a *process id*, *priority*, *arrival time*, and *duration*. The *duration* of a process is the amount of time it takes to completely execute the process. The system keeps a priority queue to keep arriving processes and prioritize the execution of processes. When a process arrives, it is inserted into the priority queue. Then, each time the system is ready to execute a process, the system removes a process with the ***smallest priority*** from the priority queue and executes it for the *duration* of the process. Once the system starts executing a process, it finishes the execution of the process completely without any interruption.
>
> Suppose that a process *p* with a very large *priority* arrives at the system while the system is executing another process. While *p* is waiting in the queue, another process *q* with a smaller *priority* arrives. After the execution of the current process is finished, the system will remove and execute *q* (because *q* has a smaller *priority*), and *p* must wait until *q* finishes. If another process with a smaller *priority* arrives while *q* is being executed, *p* will have to wait again after *q* is completed. If this is repeated, *p* will have to wait for a very long time. One way of preventing this is as follows: If a process has waited longer than a predetermined maximum wait time, we update the *priority* of the process by decreasing the priority by one. For example, if the current priority of the process is 5, then it is updated to 4. Technically, we have to perform this update at each (logical) time. However, to make the simulation program simple, we will do this update only when the system finishes the execution of a process.

For the purpose of this simulation, we assume the following:

- We use the priority queue implemented in Java's *PriorityQueue*.
- Each entry in the priority queue keeps a *process* object, which represents a process.
- Each process must have, at the minimum, the following attributes:

  pr: Integer // priority of the process

id: integer // process id
arrivalTime: integer // the time when the process arrives at the system
duration: integer // execution of the process takes this amount of time

The simulation program uses a logical time to keep track of the simulation process and the same logical time is used to represent the *arrivalTime* and *duration*. The simulation goes through a series of iterations and each iteration represents the passage of one logical time unit (in what follows we will use *time unit* to refer to *logical time unit*). At the beginning, the current time is set to time 0. Each iteration implements what occurs during one time unit and, at the end of each iteration, the current time is incremented.

The following describes the general behavior of the simulation program:

- All processes are stored in a certain data structure *D*, which is supposed to be external to the computer system.
- In each iteration (or during one time unit), the following occurs (not necessarily in the given order):
  - We compare the current time with the arrival time of a process with the earliest arrival time in *D*. If the arrival time of that process is equal to or smaller than the current time, we remove the process from *D* and insert it into the priority queue *Q* (this represents the arrival of a process at the system).
  - If no process is being executed at this time and there is at least one process in *Q*, then a process with the *smallest priority* is removed from *Q* and executed, and the wait time of the process is calculated (this will be used later to calculate the average wait time).
  - When there is more than one process with the same *smallest priority*, the Java's *PriorityQueue* will arbitrarily choose the one to be removed.
  - The wait time of a process *p* is the amount of time *p* has waited in *Q*. It is calculated by subtracting the arrival time of *p* from the time when *p* is removed from *Q*. For example, if the arrival time of *p* is 12 and *p* is removed from *Q* at time 20, then the wait time is 20 – 12 = 8.
  - If the currently running process is completed, then we update the *priorities* of some processes. The update is performed as follows. If there is any process in *Q* that has been waiting longer than the maximum wait time, then the *priority* of that process is decreased by one.
  - The current time is increased by one time unit.
- The above is repeated until *D* is empty. At this time, all processes have arrived at the system. Some of them may have been completed and removed from *Q* and some may still wait in *Q*.
- If there are any remaining processes in *Q*, these processes are removed and executed one at a time. Again, a process with the smallest priority is removed and executed first. Also, whenever the system finishes the execution of a process, priorities of processes, which have been waiting longer than max. wait time, must be updated.

A pseudocode of the simulation is given below. Note that this pseudocode is a high-level description and you must determine implementation details and convert the pseudocode to a Java program. In the pseudocode, *Q* is a priority queue. There is a Boolean variable *running* in the

pseudocode. It is used to indicate whether the system is currently executing a process or not. It is ***true*** if the system is currently executing a process and ***false*** otherwise.

```
Read all processes from an input file and store them in an appropriate data structure, D
Initialize currentTime
running = false
create an empty priority queue Q

// Each iteration of the while loop represents what occurs during one time unit
// Must increment currentTime in each iteration
While D is not empty // while loop runs once for every time unit until D is empty
    Get (don't remove) a process p from D that has the earliest arrival time
    If the arrival time of p <= currentTime
        Remove p from D and insert it into Q
    If Q is not empty and the flag running is false
        Remove a process with the smallest priority from Q
        Calculate the wait time of the process
        Set a flag running to true
    If currently running process has finished
        Set a flag running to false
        Update priorities of processes that have been waiting longer than max. wait time
End of While loop

// At this time all processes in D have been moved to Q.
// Execute all processes that are still in Q, one at a time.
// When the execution of a process is completed, priorities of some processes
// must be updated as needed.

While there is a process waiting in Q
    Remove a process with the smallest priority from Q and execute it

 Calculate average wait time
```

An input file stores information about all processes. The name of the input file is *process_scheduling_input.txt*. A sample input file is shown below (this sample input is posted on Blackboard):

```
1 4 25 10
2 3 15 17
3 1 17 26
4 9 17 30
5 10 9 40
6 6 14 47
7 7 18 52
8 5 18 70
9 2 16 93
10 8 20 125
```

Each line in the input file represents a process and it has four integers separated by a space(s). The four integers are the *process id*, *priority*, *duration*, and *arrival time*, respectively, of a process. Your program must read all processes from the input file and store them in an appropriate data structure. You can use any data structure that you think is appropriate.

While your program is performing the simulation, whenever a process is removed from the priority queue (to be executed), your program must display information about the removed process. Your program also needs to print other information as shown in the sample output below. After your program finishes the simulation of the execution of all processes, it must display the average waiting time of all processes. A sample output is shown below, which is the expected output for the above sample input.

Note that:

- This output was generated using 30 as the maximum wait time. A different maximum wait time will result in a different output.
- Your output may not be exactly the same as the one shown below. This is because when two processes have the same smallest priority, the choice is made by Java arbitrarily when a process is removed from the queue.

// print all processes first

```
Id = 1, priority = 4, duration = 25, arrival time = 10
Id = 2, priority = 3, duration = 15, arrival time = 17
Id = 3, priority = 1, duration = 17, arrival time = 26
Id = 4, priority = 9, duration = 17, arrival time = 30
Id = 5, priority = 10, duration = 9, arrival time = 40
Id = 6, priority = 6, duration = 14, arrival time = 47
Id = 7, priority = 7, duration = 18, arrival time = 52
Id = 8, priority = 5, duration = 18, arrival time = 70
Id = 9, priority = 2, duration = 16, arrival time = 93
Id = 10, priority = 8, duration = 20, arrival time = 125

Maximum wait time = 30

Process removed from queue is: id = 1, at time 10, wait time = 0 Total wait time = 0.0
Process id = 1
       Priority = 4
       Arrival = 10
       Duration  = 25
Process 1 finished at time 35

Update priority:

Process removed from queue is: id = 3, at time 35, wait time = 9 Total wait time = 9.0
Process id = 3
       Priority = 1
       Arrival = 26
       Duration  = 17
Process 3 finished at time 52

Update priority:
PID = 2, wait time = 35, current priority = 3
PID = 2, new priority = 2

Process removed from queue is: id = 2, at time 52, wait time = 35 Total wait time = 44.0
Process id = 2
```

```
       Priority = 2
       Arrival = 17
       Duration  = 15
Process 2 finished at time 67

Update priority:
PID = 4, wait time = 37, current priority = 9
PID = 4, new priority = 8

Process removed from queue is: id = 6, at time 67, wait time = 20 Total wait time = 64.0
Process id = 6
       Priority = 6
       Arrival = 47
       Duration  = 14
Process 6 finished at time 81

Update priority:
PID = 5, wait time = 41, current priority = 10
PID = 5, new priority = 9
PID = 4, wait time = 51, current priority = 8
PID = 4, new priority = 7

Process removed from queue is: id = 8, at time 81, wait time = 11 Total wait time = 75.0
Process id = 8
       Priority = 5
       Arrival = 70
       Duration  = 18
Process 8 finished at time 99

Update priority:
PID = 4, wait time = 69, current priority = 7
PID = 4, new priority = 6
PID = 5, wait time = 59, current priority = 9
PID = 5, new priority = 8
PID = 7, wait time = 47, current priority = 7
PID = 7, new priority = 6

Process removed from queue is: id = 9, at time 99, wait time = 6 Total wait time = 81.0
Process id = 9
       Priority = 2
       Arrival = 93
       Duration  = 16
Process 9 finished at time 115

Update priority:
PID = 7, wait time = 63, current priority = 6
PID = 7, new priority = 5
PID = 4, wait time = 85, current priority = 6
PID = 4, new priority = 5
PID = 5, wait time = 75, current priority = 8
PID = 5, new priority = 7

Process removed from queue is: id = 7, at time 115, wait time = 63 Total wait time = 144.0
Process id = 7
       Priority = 5
       Arrival = 52
       Duration  = 18

D becomes empty at time 125

Process 7 finished at time 133
```

```
Update priority:
PID = 4, wait time = 103, current priority = 5
PID = 4, new priority = 4
PID = 5, wait time = 93, current priority = 7
PID = 5, new priority = 6

Process removed from queue is: id = 4, at time 133, wait time = 103 Total wait time = 247.0
Process id = 4
       Priority = 4
       Arrival = 30
       Duration  = 17
Process 4 finished at time 150

Update priority:
PID = 5, wait time = 110, current priority = 6
PID = 5, new priority = 5

Process removed from queue is: id = 5, at time 150, wait time = 110 Total wait time = 357.0
Process id = 5
       Priority = 5
       Arrival = 40
       Duration  = 9
Process 5 finished at time 159

Update priority:
PID = 10, wait time = 34, current priority = 8
PID = 10, new priority = 7

Process removed from queue is: id = 10, at time 159, wait time = 34 Total wait time = 391.0
Process id = 10
       Priority = 7
       Arrival = 125
       Duration  = 20
Process 10 finished at time 179

Update priority:

Total wait time = 391.0
Average wait time = 39.1
```

Your program must write an output to an output file named *process_scheduling_output.txt*.

Documentation

You need to prepare a documentation file, named *project_documentation.docx* or *project_documentation.pdf*, that includes the following:

- Description of all data structures you used in the program.
- Discussion:
  - Any observation you had about this project.
  - What you learned from this project.

Within the source code of your program, you must include sufficient inline comments within your program. Your inline comments must include a specification of each method in your program. A specification of a method must include at least the following:

- Brief description of what the method does
- Input parameters: Brief description of parameters and their names and types, or none
- Output: Brief description and the type of the return value of the method, or none

Deliverables

You must submit the following files:

- *ProcessScheduling.java*
- All other files that are necessary to compile and run your program

Combine all files into a single archive file and name it *LastName_FirstName_project.EXT*, where *EXT* is an appropriate file extension (such as *zip* or *rar*). Upload this file to Blackboard.

Grading

The project is worth 100 points.

Your simulation program will be tested with a test input and points will be deducted as follows:

- If your program does not compile, 60 points will be deducted.
- If your program compiles but causes runtime errors, 50 points will be deducted.
- If processes are not removed (from *Q*) in the correct order, up to 20 points will be deducted.
- If the removal times (from *Q*) are incorrect, up to 20 points will be deducted.
- If priorities are not updated correctly, up to 20 points will be deducted.
- If the average wait time is wrong (assuming there is no other error), up to 10 points will be deducted.

- If you do not include the description of data structures in your documentation, up to 10 points will be deducted.
- If the *observation* and *what you learned* in your documentation are not substantive, up to 10 points will be deducted
- If you do not include method specifications or sufficient inline comments, up to 10 points will be deducted.