CS526
Project Assignment

Due: 12/5

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.

You must use Java's built-in *PriorityQueue* or Python's built-in *PriorityQueue* (which is defined in *queue* module).

Name your program *ProcessScheduling.java* or *process_scheduling.py*. 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 *priority* of a process is an integer from 1 to 10. 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 ***largest priority*** from the priority queue and executes it for the *duration* of the process. If there are more than one process with the same largest priority, then the process with the earliest arrival time, among those with the same largest priority, must be removed from the priority queue. For example, if process *p*1 and process *p*2 have the same largest priority and process *p*2's arrival time is earlier than the arrival time process *p*1, then *p*2 is removed from the priority queue. Once the system starts executing a process, it finishes the execution of the process completely without any interruption.
>
> Technically, it would be more reasonable to perform this update at each (logical) time. However, to make this simulation program simple, we will do this update only when the system finishes the execution of a process.

In your program, you must define a *Process* class with the following attributes:

- pr // priority of a process
- id // process id
- arrival_time // the time when a process arrives at the system
- duration // execution of a process takes this amount of time

A *Process* object should store one process and you must keep *Process* objects in your priority queue.

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.

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 in the system at this time and there is at least one process in *Q*, then a process with the *largest 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). If there are more than one process with the same largest priority, then the process with the earliest arrival time, among those with the same largest priority, must be removed from the priority queue.
  - 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.
  - 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 largest priority is removed and executed first.

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 or Python 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 update currentTime in each iteration appropriatly
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 currently running process has finished
Set a flag *running* to false
If *Q* is not empty and the flag *running* is false
Remove a process with the largest priority from *Q*
(tie is broken by arrival time)
Calculate the wait time of the process
Set a flag *running* to true

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 largest 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 26
2 3 15 10
3 1 17 65
4 3 17 17
5 1 9 39
6 6 14 125
7 5 18 30
8 6 18 25
9 5 16 52
10 5 20 20
```

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 *D*. 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 display 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.

```
Id = 1, priority = 4, duration = 25, arrival time = 26
Id = 2, priority = 3, duration = 15, arrival time = 10
Id = 3, priority = 1, duration = 17, arrival time = 65
Id = 4, priority = 3, duration = 17, arrival time = 17
Id = 5, priority = 1, duration = 9, arrival time = 39
Id = 6, priority = 6, duration = 14, arrival time = 125
Id = 7, priority = 5, duration = 18, arrival time = 30
Id = 8, priority = 6, duration = 18, arrival time = 25
Id = 9, priority = 5, duration = 16, arrival time = 52
Id = 10, priority = 5, duration = 20, arrival time = 20


Process removed from queue is: id = 2, at time 10, wait time = 0 Total wait time = 0.0
Process id = 2
       Priority = 3
       Arrival = 10
       Duration  = 15
Process 2 finished at time 25

Process removed from queue is: id = 8, at time 25, wait time = 0 Total wait time = 0.0
Process id = 8
       Priority = 6
       Arrival = 25
       Duration  = 18
Process 8 finished at time 43

Process removed from queue is: id = 10, at time 43, wait time = 23 Total wait time = 23.0
Process id = 10
       Priority = 5
       Arrival = 20
       Duration  = 20
Process 10 finished at time 63

Process removed from queue is: id = 7, at time 63, wait time = 33 Total wait time = 56.0
Process id = 7
       Priority = 5
       Arrival = 30
       Duration  = 18
Process 7 finished at time 81

Process removed from queue is: id = 9, at time 81, wait time = 29 Total wait time = 85.0
Process id = 9
       Priority = 5
       Arrival = 52
       Duration  = 16
Process 9 finished at time 97

Process removed from queue is: id = 1, at time 97, wait time = 71 Total wait time = 156.0
Process id = 1
       Priority = 4
       Arrival = 26
       Duration  = 25
Process 1 finished at time 122

Process removed from queue is: id = 4, at time 122, wait time = 105 Total wait time = 261.0
Process id = 4
       Priority = 3
       Arrival = 17
       Duration  = 17
```

```
D becomes empty at time 125

Process 4 finished at time 139

Process removed from queue is: id = 6, at time 139, wait time = 14 Total wait time = 275.0
Process id = 6
       Priority = 6
       Arrival = 125
       Duration  = 14
Process 6 finished at time 153

Process removed from queue is: id = 5, at time 153, wait time = 114 Total wait time = 389.0
Process id = 5
       Priority = 1
       Arrival = 39
       Duration  = 9
Process 5 finished at time 162

Process removed from queue is: id = 3, at time 162, wait time = 97 Total wait time = 486.0
Process id = 3
       Priority = 1
       Arrival = 65
       Duration  = 17
Process 3 finished at time 179

Total wait time = 486.0
Average wait time = 48.6
```

In addition to displaying the output on the screen, your program must also write an output to an output file named *process_scheduling_output.txt*.

This assignment is an individual work. You must not cooperate with other students.

Documentation

You need to prepare a documentation file, named *project_documentation.docx* or *project_documentation.pdf*, which 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/function in your program. A specification of a method/function must include at least the following:

- Brief description of what the method/function 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/function, or none

Deliverables

You must submit the following files:

- Documentation file described above
- *ProcessScheduling.java* or process_scheduling.py
- 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, up to 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 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/function specifications or sufficient inline comments, up to 10 points will be deducted.