CMPT330 – Concepts of Operating Systems, Winter 2024
Laboratory Demonstration Three, March 22, 2024

Create the start of a CPU Simulator in Java that creates processes and puts them in a ready queue. Use two classes to do this: Process.java and Simulator.java

Process.java – this class encapsulates one process. It has (at least) the following data members and methods

- type – a name of what kind of process this is
- pid – a process ID, assigned sequentially where each process is assigned the next PID. No two processes should have the same PID.
- burst – how long this process runs before it will block for I/O
- ioBlock – how long this process will take while waiting for I/O
- cpuTime – how long in the CPU this process needs before completing
- creationTime – a time stamp of when this process was created (not really a Time object or a date)
- a constructor – requires at least the type, burst time, ioBlock time, cpuTime, and the current time of the system.
- toString – returns the process information as a String

You may need additional data members and methods to complete the assignment, but these should do for the lab. You may add more parameters to the constructor if you like, but it shouldn't be required. Remember that data members should be declared as private.

Simulator.java – this class is the start of the simulator that you can also use in assignment three.

Your Simulator class should start a loop and randomly create two types of processes:

Interactive:	**Batch**:
Type: "Interactive" (a String)	Type: "Batch" (a String)
Burst: 10 + / - 20%	Burst: 250 + / - 20%
CPU Time: 20 + / - 20%	CPU Time: 500 + / - 20%
IO Block Time: 5 + / - 20%	IO Block Time: 10 + / - 20%
Creation Time Stamp: time of creation	Creation Time Stamp: time of creation

Write a method to call from your loop that creates these processes. Each time the loop in Simulator runs there should be a 10% chance of creating an interactive process, and a 10% chance of creating a batch process. It is possible to have no processes created, one interactive process, one batch process, or two processes (one of each). Your method should return the processes back to the main loop. Each time a process is returned, add it to a ready queue. You may create your own data structure to implement the ready queue or use a data structure built into Java. Keep in mind, however, that you don't have a maximum size of the ready queue.

Note that your burst, CPU time, and IO Block Time should be randomly assigned as specified above. For example, the interactive burst time should be 10 +/- 20% meaning the burst time could be 8, 9, 10, 11, or 12.

Have your Simulator loop until both types of processes are created at the same time. Then print out all the processes in your ready queue to the console and exit the program.

Shown below are two processes created at the same time. It is likely you will have other processes created before two are created at once.

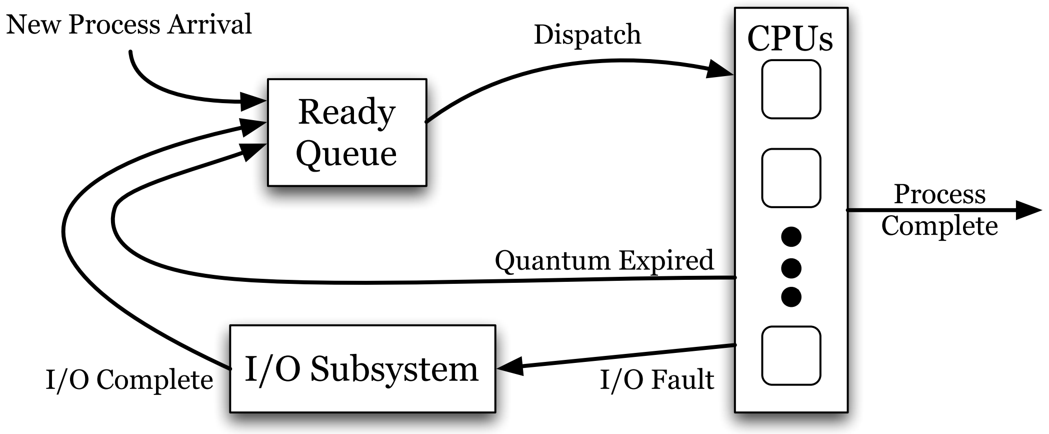
```
Process:
Type: Interactive
PID: 0
Burst: 8
IO Block Time: 4
CPU Time: 16
Creation Time Stamp: 14

Process:
Type: Batch
PID: 1
Burst: 261
IO Block Time: 10
CPU Time: 596
Creation Time Stamp: 14
```

Remember to include suitable comments above every method or class, and to add descriptive comments to explain your program.

Note that since your processes are placed in the ready queue in order of creation, they should have sequential process ID numbers when printed (FIFO).

After you have written your solution, demonstrate it to the instructor, and upload your source and executable files to Moodle in a single zip file.

Shown below is a Simulator diagram of which you are writing the “New Process Arrival” and the “Ready Queue” for this lab.


New Process Arrival
Ready Queue
Dispatch
CPUs
Process Complete
Quantum Expired
I/O Complete
I/O Subsystem
I/O Fault