Processor Execution Simulator

Assignment Description

In this assignment, you are required to build a **simulator** that simulates processor execution for processes (i.e., tasks). Assuming that we have P processors, consider the following notes about the processor execution:

- A processor can only execute one task at a time.
- The number of processors is fixed during the whole simulation.
- All processors are synchronized, i.e., they all have the same clock.
- Processors are given unique ids as follows: P_1, P_2, P_3, P_4, etc.
- Tasks are given unique ids as follows: T_1, T_2, T_3, T_4, etc.
- Clock cycles are given unique ids as follows: C_1, C_2, C_3, C_4, etc.
- A task is defined by the following:
  - Creation time: the clock cycle in which the task was created.
  - Requested time: the number of clock cycles the task needs to execute till completion.
  - Completion time: the clock cycle in which the task was completed.
  - Priority: which can be either high or low.
  - State: which can be either one of three states (see below).
- When a task is created, it must be automatically put in a queue.
- The simulator has a scheduler that is responsible for determining the assignment of tasks that are waiting in the queue to processors that are available.
- A task can have only one of three states: either “waiting” in the queue, “executing” on a processor, or “completed”.
- A processor can have only one of two states: “busy” or “idle”.
- If there is a processor that is available, then the schedular must prioritize the assignment of tasks with “high priority” over tasks with “low priority”.
- If there is a tie, i.e., a processor became available and there are two “high priority” tasks waiting, then I will leave it to you (the student) to determine how the scheduler should break ties. The same goes for breaking ties between “low priority” tasks.
- Low priority tasks can be interrupted. For example, if a high priority task is created, and there was a processor that is running a low priority task, then the schedular must remove the low priority task from the processor and put it in the waiting queue, and then assign the high priority task to the processor.
- If a low priority task is interrupted, when executed again, it only needs to run for the remaining time of its requested time. For example, if a low priority task requested to run for 30 cycles, and it was interrupted during the tenth cycle, then when it runs again, it only needs to run for 20 cycles.
- High priority tasks cannot be interrupted.
- For simplicity, we will assume task creation time is negligible. For example, if a task was created during cycle N, and there was an available processor, then this task can be immediately assigned to this processor in the same cycle N.

Implementation requirements

- Implement the simulator with the above specifications using object-oriented programming
- What should be a class? What interface should each class have? What are the relationships between classes? What data structures should I use? These are some of the questions that you should carefully consider.
- Assume that your simulator is maybe extended by others in the future. Therefore, pay careful attention to object-oriented design to make your code: organized, flexible, and reusable.
- First, implement your simulator while assuming a naïve tie breaking strategy for tasks. Second, consider implementing advanced tie breaking strategies. The reason the tie breaking strategy is important because it can significantly impact performance, i.e., how long it takes all processors to executed all tasks. However, your main concern should be first to implement the full simulator with the given specifications. Then, worry about the tie breaking strategy.
- Your simulator must be configurable using an input configuration file that specifies the number of processors and tasks, as well as the needed descriptions for each task (creation time, requested time, and priority). I will leave it to you to determine the exact nature of the configuration file.
- Your simulator must produce meaningful output to the user that contains information about the total simulation time, completion times for all tasks, and other information you think is important. I will also leave it to you to determine the exact nature of the simulator output.

Deliverables

1. Source code
2. Runnable jar file with suggested input configuration files.
3. An organized report (in pdf format) that contains an explanation of your software design (with a UML diagram), and used data structures and algorithms in your code. Make sure to also describe any particular strengths or interesting highlights of your work.
4. A URL of an unlisted youtube video that contains a **demonstration** of your simulator. At the beginning of the video, make sure to introduce yourself, and state your education and experience. Also, make sure to show your face during the whole video. You do not need to describe your classes or algorithms in the video. Preferred video time is 5-8 minutes.