Assignment Three

In this assignment, you will implement a task scheduler in Java.

Background

An embedded system is a computer system performing dedicated functions within a larger mechanical or electrical system. Embedded systems range from portable devices such as Google Glasses, to large stationary installations like traffic lights, factory controllers, and complex systems like hybrid vehicles, and avionic. Typically, the software of an embedded system consists of a set of tasks (threads) with timing constraints. Typical timing constraints are release times and deadlines. A release time specifies the earliest time a task can start, and a deadline is the latest time by which a task needs to finish. One major goal of embedded system design is to find a schedule for the task set such that all the timing constraints are satisfied.

Task scheduler

We assume that the hardware platform of the target embedded systems is a single processor with m identical cores. The task set $V=\{v_1, v_2, \dots, v_n\}$ consists of n independent tasks. The execution time of each task is one time unit. Each task v_i (i=1, 2, ,,, n) has a release time r_i and a deadline d_i. All the release times and deadlines are non-negative integers. You need to design an algorithm for the task scheduler and implement it in Java. Your task scheduler uses EDF (Earliest Deadline First) strategy to find a feasible schedule for a task set. A schedule of a task set specifies when each task starts and on which core it is executed. A feasible schedule is a schedule satisfying all the release time and deadline constraints.

The EDF strategy works as follows.

- At any time t, among all the ready tasks, find the one with the smaller deadline, and schedule it on an idle core. Ties are broken arbitrarily. A task v_i (i=1, 2, ,,, n) is ready at a time t if $t \geq r_i$ holds.

We can prove that the EDF strategy is guaranteed to find a feasible schedule whenever one exists for a set of independent tasks with unit execution time.

An Example

Consider a set of 10 independent tasks whose release times and deadlines are shown in Table 1. The target processor has 3 identical cores. A feasible schedule of the task set is shown in Figure 1.

Table 1: A set of tasks with individual release times and deadlines.

Task	Release time	Deadline
v_1	0	4
v_2	1	2
v_3	1	2
v_4	1	2
v_5	1	3
v_6	4	7
v_7	4	6
v_8	4	6
v_9	4	7
v_{10}	5	6

In this example, if the number of cores is 2, no feasible schedule exists. The reason is that one of the tasks v_2, v_3 and v_4 must miss its deadline.

Core 1	v_1	v_2	v_5		v_6	v_{10}	
Core 2		v_3			v_7	v_9	
Core 3		v_4			v_8		

Time 0 1 2 3 4 5 6 7

Figure 1: A feasible schedule.

The TaskScheduler class

You need to write a task scheduler class named **TaskScheduler**. A template of the **TaskScheduler** class is shown as follows.

```
public class TaskScheduler
{
  …
  static void scheduler(String file1, String file2, Integer m) {};
  …
}

class ClassX {}  /** Put all the additional classes you need here */
…
```

You can define any fields, constructors and methods within the TaskScheduler class. You can also define additional classes. You must put all the additional classes in the file Taskscheduler,java without class any class modifiers.

The main method scheduler(String file1, String file2, Integer m) gets a task set from file1, constructs a feasible schedule for the task set on a processor with m identical cores by using the EDF strategy, and write the feasible schedule to file2. If no feasible schedule exists, it displays " No feasible schedule exists" on the screen. This method needs to handle all the possible cases properly when reading from file1 and writing to file2. All the possible cases are as follows.

1. file1 does not exist.
2. file2 already exists.
3. The task attributes (task name, release time and deadline) of file1 do not follow the format as shown next.

Both file1 and file2 are text files. files1 contains a set of independent tasks each of which has a name, a release time and a deadline. A task name is a string of letters and numbers. All the release times are non-negative integers, and all the deadlines are natural numbers. The format of file1 is as follows.

v_1 r_1 d_1 v_2 r_2 d_2 … v_n r_n d_n

Two adjacent attributes (task name, release time and deadline) are separated by one or more white space characters. A sample file1 is shown here.

For simplicity, you may assume that all the task names are distinct.

file2 has the following format.

v_1 t_1 v_2 t_2 … v_n t_n

where t_i is the start time of the task v_i (i=1, 2, …, n). In file2, all the tasks must be sorted in non-decreasing start times. A sample file2 is shown here. Notice that you do not need to include the core on which each task is executed.

Time complexity requirement

The time complexity of your scheduler is required to be no higher than O(m*n log n), where n is the number of tasks, and m is the number of cores (**Hints: use a fast sorting algorithm and the priority queue using heap**). **You need to include the time complexity analysis of your task scheduler in the TaskScheduler class file as comments.** There is no specific requirement on space complexity. However, try your best to make your program space efficient.

How to submit your code?

Login to your CSE account, and type the following command.

```
give cs9024 assn3 TaskScheduler.java
```

Marking

Marking is also based on the correctness and efficiency of your code. Your code must be well commented.
The full mark of this assignment is 7 if the time complexity of your scheduler satisfies the above-mentioned requirement. Otherwise, the full mark will be reduced. For example, if the time complexity of your scheduler is $O(m*n^2)$, the full mark is 5.

Deadline

The deadline is 11:59:59 pm, 12 Oct.