Assignment Three

This is an individual assignment. In this assignment, you will implement a task scheduler in Java using heap-based priority queues.

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 feasible 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, Core1, Core2, …, Corem. The task set V={v_1, v_2, …, v_n} consists of n independent, non-pre-emptible tasks. A non-pre-emptible task cannot be pre-empted by another task during its execution, i.e., it runs until it finishes. Each task v_i (i=1, 2, …, n) has four attributes: a unique task name v_i, an execution time c_i, a release time r_i, and a deadline d_i (d_i>r_i). All the execution times, the release times and deadlines are non-negative integers. You need to design a 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 problem of finding a feasible schedule for this task scheduling problem is NP-complete. It is widely believed that an NP-complete problem has no polynomial-time algorithm. However, nobody can prove it.

First, we introduce two definitions: scheduling point and ready task.

- A scheduling point is a time point at which a task is scheduled on a core.
- A task v_i (i=1, 2, ,,, n) is ready at a time t if $t \geq r_i$ holds.

The EDF scheduling strategy works as follows:

- At each scheduling point t_i ($t_i \leq t_i+1$, i=1, 2, …), among all the ready tasks, find a task with the smallest deadline, and schedule it on an idle core such that its start time is minimized. Ties are broken arbitrarily.

Since this task scheduling problem is NP-complete, the EDF strategy is used as a heuristic which is not guaranteed to find a feasible schedule even if a feasible schedule exists.

Example One

Consider a set S_1 of 6 independent tasks whose release times and deadlines are shown in Table 1. The target processor has two identical cores. A feasible schedule of the task set by using EDF scheduling strategy is shown in Figure 1.

Table 1: A set S_1 of 6 tasks with individual release times and deadlines

Task	Execution time	Release time	Deadline
v_1	4	0	4
v_2	4	1	5
v_3	5	3	10
v_4	6	4	11
v_5	4	6	13
v_6	5	6	18

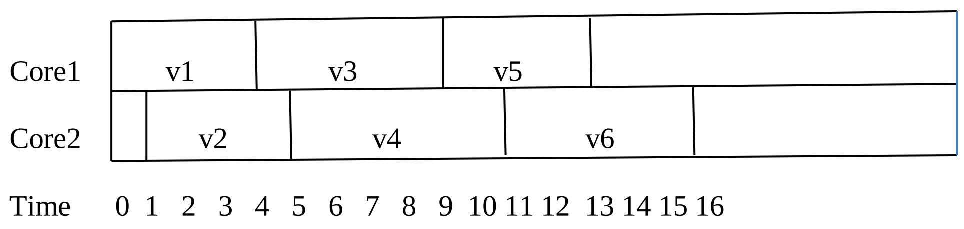

Figure 1: A feasible schedule for S_1

Example Two

Consider a set S_2 of 6 independent tasks whose release times and deadlines are shown in Table 2. The target processor has two identical cores. A schedule of the task set by using EDF scheduling strategy is shown in Figure 2. As we can see, in the schedule, v_6 finishes at time 16 and thus misses its deadline. Therefore, the schedule is not feasible. However, a feasible schedule, as shown in Figure 3, does exist.

Table 2: A set of tasks with individual release times and deadlines

Task	Execution time	Release time	Deadline
v_1	4	0	4
v_2	4	1	5
v_3	5	3	10
v_4	6	4	11
v_5	4	9	16
v_6	5	10	15

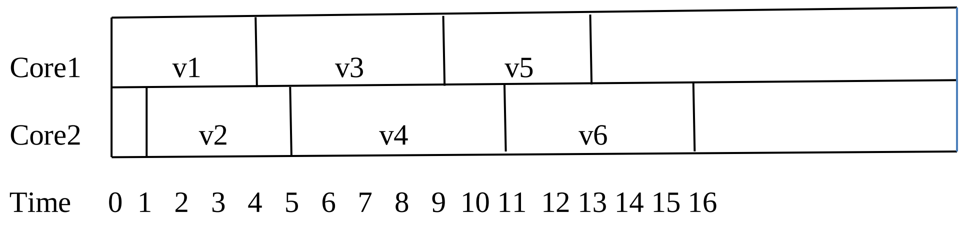

Figure 2: An infeasible schedule constructed by the EDF scheduling strategy

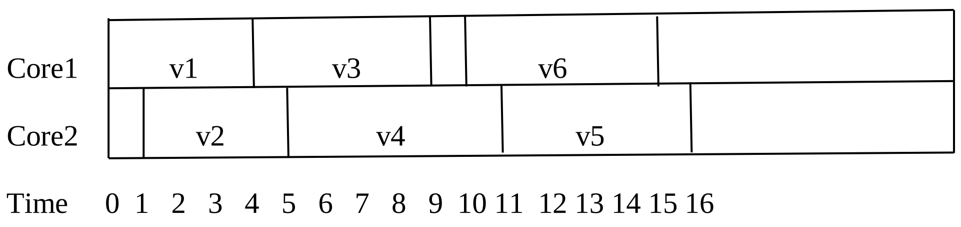

Figure 2: A feasible schedule for S_2

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, int 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, int 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.

Both file1 and file2 are text files. files1 contains a set of independent tasks each of which has a name, an execution time, a release time and a deadline in that order. A task name is a string of letters and numbers starting with a letter. Each of the execution times, release times and the deadlines is a string of digits between 0 and 9. All the release times are non-negative integers, and all the execution times and the deadlines are natural numbers. The format of file1 is as follows:

$v_1\ c_1\ r_1\ d_1\ v_2\ c_2\ r_2\ d_2 \ldots v_n\ c_n\ r_n\ d_n$

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

For simplicity, you may assume that all the task names in file1 are distinct.

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. In this case, print "file1 does not exist" and the program terminates.
2. file2 already exists. In this case, overwrite the old file2.

3. The task attributes (task name, release time and deadline) of file1 do not follow the formats as shown before. In this case, print “input error when reading the attribute of the task X” and the program terminates, where X is the name of the task with an incorrect attribute.

file2 has the following format:

$v_1\ p_1\ t_1\ v_2\ p_2\ t_2\ \ldots\ v_n\ p_n\ t_n$

where each v_i (i=1, 2, …, n) is the task name, p_i is the name of the core where v_i is scheduled, and t_i is the start time of the task v_i in the schedule. In file2, all the tasks must be sorted in non-decreasing order of start times. A sample file2 is shown here.

Time complexity requirement

You need to include your time complexity analysis as comments in your program. The time complexity of your scheduler is required to be no higher than O(n*log n), where n is the number of tasks (**Hints: use heap-based priority queues**). **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.

net.datastructures-4-0 package

You can use net.datastructures-4-0 where there is a heap-based priority queue class named HeapPriorityQue.java. Read through its code and understand how it works.

Restrictions

All the other data structures and algorithms that are not in net.datastructures-4-0 must be implemented in the TaskScheduler class. You are NOT allowed to use any sorting algorithms and priority queues provided by Java.

How to submit your code?

Follow this link: https://cgi.cse.unsw.edu.au/~give/Student/give.php. Do the following:

1. Use your z-pass to log in.
2. Select current session, COMP9024 and assn3.
3. Submit 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 0.

Deadline

The deadline is **11:59:59 pm, 14 May**. No late submission will be accepted.