ITCS 6010/8010 **Real-Time Embedded Systems**

Homework 4

Each student must solve this assignment **individually**. You must submit your files electronically via Moodle. You are required to submit program files (no executable files or project files, just the actual code) and a simple README file giving any instructions needed to build and run your program. All submission files must be **zipped** and a single file must be submitted. You are free to use any programming language for this assignment (e.g., C++, Java, MATLAB, Python, etc.).

Note: This assignment is not applicable to students who have discussed relevant alternate projects with me.

(50 points) Design and implement a scheduling simulator that supports the Earliest Deadline First (**EDF**) scheduling policy in conjunction with the Priority Inheritance Protocol (**PIP**). An event-driven approach is recommended for purposes of efficiency. However, you could use a simpler, but less efficient, time-driven approach. The input to your simulator will be provided through a text file that your program must parse. You may assume *integral* time units (i.e., there is no need to support fractions of a time unit).

If two jobs have the same absolute deadline, assign higher priority to the job that was released earlier. If both jobs additionally have the same release time, assign higher priority to the job belonging to the task listed earlier in the input file. If a job misses its deadline, abort it at the deadline and continue the simulation. Indicate resource acquisition and release as shown in the example output. You may assume that an *ordering* is imposed on resource requests in order to *avoid* deadlocks. So, no deadlock detection or recovery is needed.

Input file format

```
<simulation stop time>

<number of tasks>

for each task

    <name>   <ϕ>   <P>   <e >   <D>  <num resource regions>

    for each resource region

        <resource name>    <from>    <length of region>
```

Example

Input:

```
7

4

A    6    35    3    28    1    R2  1  2

B    4    30    4    30    1    R1   2  1

C    0    45    7    40    2    R1   1   3    R2    2    2

D    3    40    8    35    2    R1   1   4    R2    2    2
```

Output:

```
Time  Job   Actions                    Current priority

  0   C1

  1   C1    lock R1

  2   C1    lock R2

  3   D1

  4   B1

  5   B1

  6   C1    unlocks R2, R1             B1

  7   B1    locks R1, unlocks R1
```

Note:

1. *Job numbers start from 1.*
2. *All "lock" actions occur at the beginning of the specified unit of time and "unlock" actions occur at the end of the specified unit of time. So, C1 locks R1 at the beginning of time unit 1 and unlocks R1 at the end of time unit 6. Similarly, B1 locks R1 at the beginning of time unit 7 and unlocks it at the end of the same time unit.*
3. *The current priority field is only required if the current priority of a job is not equal to its original priority. In this case, you must indicate the task name + job number of the job whose priority the current job inherits in a given unit of time.*