CS 346 – Lab 7 – Real-time Scheduling

Today we will experiment with two classic real-time scheduling methods: Earliest Deadline First (EDF) and Rate-Monotonic (RM) scheduling. Create a directory called lab07 and do all your work there. Inside the directory ~chealy/www/lab07 I have a couple of Java source files that you may find useful in your implementation.

To make today's work manageable, let's agree on some assumptions: In a task set, all jobs are initially ready to execute, and no new jobs enter the system during execution. All job periods are positive integers.

Your program should have enough output to make it clear to the user what the job schedules are under both EDF and RM. A possible Java organization to your program is as follows.

- Job class – provided to you
  - Attributes of name, period, execution time, beginning of period, deadline
  - Initial-value constructor
  - getName()
  - getPeriod()
  - getExecTime()
  - setPeriodBegin()
  - setDeadline()
  - getPeriodBegin()
  - getDeadline()
  - toString()

- JobPeriodComparator – provided to you

- Driver/main class – you need to write
  - Create a set of jobs. As an example, you can refer to an example we did in class with 3 jobs such as (3/10, 4/12, 5/15). In the long run, you might want to declare another class that incorporates a collection of jobs: for this set of jobs you can compute its two schedules, its total CPU utilization, and whether RM is successful.

  - Find the hyperperiod: the least common multiple of the job periods. You can write a separate method to compute this.

  - Compute the CPU utilization for the task set.

  - Compute the EDF schedule up to the hyperperiod.
    - At each point in time, you need to detect if there are no jobs available. In this case, you need to keep the CPU idle until one job begins its period.

    - **Output**: Each time you schedule a job, indicate which job it is and during what time interval it executes. Also indicate intervals of idle time. You may also find it helpful to print each job's toString() at the points in time when your program needs to decide which job to execute next.

  - Compute the RM schedule up to the hyperperiod.
    - You need to sort the jobs according to their period. Note that because of this priority system, jobs may be interrupted by a higher priority job. This will happen

automatically as you consider jobs individually in descending order of priority, and make sure you don’t schedule two jobs at the same time.

    - You also need to detect a deadline miss, in which case the task set is not schedulable.

    - **Output**: For each single time unit up to the hyperperiod, indicate which job is executing. If a deadline is missed, the program should indicate which job missed a deadline and when.

  - Test your program on this set of jobs: (4/10, 4/15, 10/35). Does RM work?

  - As we did in class, play with different job parameters (i.e. adjust the execution time and/or period) to see how much CPU utilization causes the RM schedule to succeed or fail in the long run.

That’s all! When you are done, please turn in any source files that you modified.