

INFO-F-404: Real-Time Operating Systems

2013 – 2014 Project 1: Least Laxity First

DELHAYE Quentin

October 28, 2013

1 Implementation

1.1 Task Generator

The `main` function of `taskGenerator` does the following:

- parsing the program arguments;
- creating an `Generator` object with those arguments;
- asking the object to generate the tasks and saving them to a file.

The WCET of each task is randomly chosen in the interval $[1; 30]$. Similarly, the deadline randomly chosen between the end of the WCET and the end of the period.

Saving the tasks into a file is optional, since the analyzer will just require a table with the parameters.

1.2 LLF Scheduler

The `main` function of `simLLF` does the following:

- parsing the program arguments;
- extract the information concerning the tasks from the file given in argument;
- create a new object `Simulator` that will handle the rest of the job.

The simulator begins by creating an array of `Task` objects with the required parameters. The simulation can then begin, for more information about this part, please refer to the appendix A.

The computation of the study interval is based on the well-known Euclidean algorithm that give the *gcd* of two numbers. Knowing that $\text{gcd}(a,b,c) = \text{gcd}(\text{gcd}(a,b),c)$, the *lcm* of the set is immediate.

Since the system has to be scheduled following the LLF, the `setPriorities` method will ask to each to compute its own laxity, and then will sort the array containing all the tasks and their corresponding laxity.

When the simulation is successful, that is if no task has ever missed its deadline, the `Simulator` class creates then a new `GraphCreator` object that will create a `png` image representing the scheduling of the system. This image is created using the `pngwriter` library (which should then be installed on the system). The points used by this object to generate the image are the beginning and ending time of each job, the beginning and ending type (preemption, recovery, or normal) as well as the task number (an idle time having an `id = -1`).

As an example, the figure 1 represents the scheduling of the system proposed in the first chapter of the course, slides 83 and 84, with a re-computation of the priorities each unit of time ($\text{delta} = 0$).

The red triangles represent a preemption, the white arrows the arriving of a new job, the white diamonds the deadline, and the green area the idle time of the CPU.

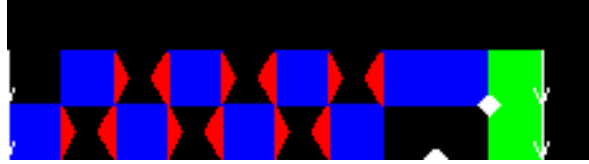


Figure 1: LLF Scheduling of two tasks

1.3 LLF Analyzer

The analyser `LLF_study` implements all the previous classes. It will begin by generating a set of tasks, and then will simulate them.

For further details, please refer to the section 3.

2 Difficulties

Split the utilization between the tasks During the generation of the tasks based on the arguments given to `taskGenerator`, one of the difficulties was to determine what should the utilization of each task be in order to have the expected system utilization at the end. The solution chosen is the simplest: assigning the same utilization to each task.

It is to be noted that due to this method and because of the use of natural numbers, the actual utilization is a bit off.

3 Simulations

There are three free parameters in our system: the utilization, the number of tasks in the system, and the *delta*¹.

They will be all compared to the other two, each time one parameter being fixed.

Notes: some cells will be filled with "N/A" in the case the system was not schedulable. Otherwise, each cell will contain a triplet of intergers: {study interval, preemptions, idle time}

3.1 Results

First round: fixed number of tasks (3), various *delta* and utilization: table 1

Second round: fixed *delta* (2), various number of task and utilization: table 2 Let's note that the second ligne of the table 2 is pathologic.

¹The *delta* is the period at which the priorities are computed.

	50	60	70	80	90
1	1008;78;519	2550;120;1172	672;20;234	N/A	N/A
2	2340;52;1201	350;0;163	N/A	N/A	N/A
5	N/A	29750;25;12521	780;23;212	N/A	N/A
10	42120;64;21815	N/A	360;2;97	N/A	N/A
20	N/A	N/A	N/A	N/A	N/A

Table 1: Horizontally: *delta*, vertically: utilization

	50	60	70	80	90
2	N/A	N/A	N/A	N/A	N/A
3	5400;97;2898	N/A	N/A	N/A	N/A
5	75600;0;39395	N/A	33810;691;10731	N/A	N/A
7	3800160;0;2106620	12870;86;5336	46200;999;14795	N/A	N/A
9	249480;1573;130734	6054750;32531;2671512	N/A	N/A	N/A

Table 2: Horizontally: number of task, vertically: utilization

	1	2	5	10	20
2	84;0;34	24;0;13	N/A	N/A	N/A
3	840;25;382	30940;62;12871	225;0;116	N/A	N/A
5	10080;468;3976	9600;600;3807	N/A	1235520;7900;486857	N/A
7	23760;90;9825	4924458;132770;1883395	N/A	N/A	N/A
9	2079000;63181;911885	41441400;175450;17708724	297000;730;127153	N/A	N/A

Table 3: Horizontally: number of task, vertically: *delta*

Third round: fixed utilization (70), various number of task and *delta*: table 3

3.2 Conclusions

Utilization An utilization of 80 percent or higher always leads to an unschedulable system. This may be explained by the approximation made on this utilization in the implementation.

delta A *delta* of 10 or more units of time will almost always lead to an unschedulable system. Indeed, some tasks may miss their opportunity to run a job and thus miss their deadline.

Preemptions It increases with the utilization and the with a lower *delta*.

Idle time The higher the utilization, the higher the ration (idle time/study interval).

A State Diagram of the simulation method

