INFO-F-404: Real-Time Operating Systems

2016 – 2017 Project 1: Global vs. Partitioned DM

1 Main goal

Study the performance of DM scheduling algorithm on systems with periodic, asynchronous tasks and constrained deadlines.
We will consider systems of n periodic, asynchronous and independent tasks $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ with constrained deadlines embedded on a multiprocessor systems. We will consider two strategies: global and partitioned.

2 Project details

For this project, we ask you to do the following parts:

Simulator

Implement an DM priority assignment and a simulator that can simulate the scheduling of a system (given as a parameter in a file) on the interval $I = [0; O_{max} + 2 \times P]$ (where P is a period of the system in case of simple scheduling with DM algorithm).

You should be able to use two strategies to assign tasks (and jobs) to processors: the global strategy and the partitioned strategy (best fit). We will consider that all processors in the system are identical i.e., a job needs the same amount of time to be executed on any processor.

You should be able to execute your program using the following command line:

```
./simDM [-g | -p] <tasksFile> <processorsNbr>
```

for example:

```
./simDM -g tasks.txt 4
```

In this case the system of tasks is described in the file `tasks.txt`, and the system contains 4 processors, the strategy to be used is "global" given by the option `-g` (`-p` denotes partitioned strategy).

Program `simDM` should be able to generate the full simulation of the system described by `tasks.txt` on its study interval. In case of partitioned strategy, each processor will have its own study interval (since it will get its own subsystem of tasks). Program `simDM` have to print at least the following information: the number of processors required, the interval I as well as number of preemptions (total and per processor), the total (and per processor) idle time during the study interval, total system utilisation and system utilisation per processor. Program `simDM` should also inform the user in case if the system could not be scheduled using the DM priority assignment with the strategy that was chosen by the user. The way that this information is displayed is left to the designer of the simulator, you a also free to output additional information on the screen or to a file. Choose wisely the kind of information that your program will output (to a file and/or on the screen).

Generator

Implement a generator of random periodic, asynchronous systems with constrained deadlines. This generator should be able to generate a system with given parameters (utilisation factor and number of tasks), for example:

`./taskGenerator -u 70 -n 8 -o tasks.txt`

have to generate a file `tasks.txt` that describes a system of 8 tasks the utilisation of this system is 70% (utilisation of the generated system does not have to be equal to 70%, but should be very close to it).

Study

Implement a program (`studyDM`) to test DM's performances. You should wisely use modules that you've already implemented as well as the kind of parameters that you submit to this program.

Comparison tests: system's load (total time spend working), number of required processors and schedulability depending on the total utilisation of the system, number of tasks in the system as well as depending on the strategy that is used to distribute the load of the system to several processors.

Report

Write a short report that contains:

1. short description of your code (diagrams) and implementation choices,
2. a section where you describe difficulties that you met during this project (and solutions that you found),
3. a part where you describe how you made the comparison tests,
4. a section that describe and discuss the result of your simulations (tables, graphics)[1].

All programs (`simDM`, `taskGenerator` and `studyDM`) have to be implemented using *C++* programming language. You are allowed to add more additional (optional) command line parameters to your programs.

We ask you to use the following file format to describe systems of tasks. Each line of a file describes one task and contains: `Offset Period Deadline WCET`.

Here is an example:

```
0 50 50 30
10 100 70 30
0 80 60 10
```

2.1 Bonus part

This part is not mandatory, but it is worth some bonus points (*only* if all other parts work).

You can write a program that generates a visual output of the scheduling. You may choose the format (pdf, png, bmp, avi, etc.).

[1] very important part of the report!

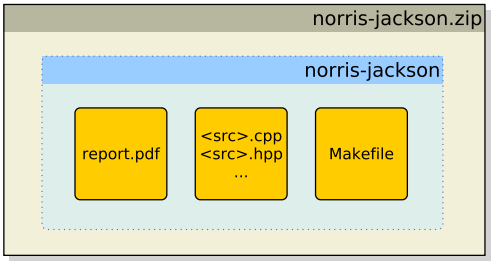

Figure 1: Example: the minimum to be submitted. Your source files might be placed in a separate folder.

3 Submission and planning

This project should be done in groups of 2 or 3, you may choose your partner(s). This project has to be submitted before 23:55:00 o'clock on the 12^{th} of December of 2015 (Monday of the last week of lectures). Your project has to work properly (compile and execute) under *Linux* in rooms NO3.007, NO4.008 and NO4.009 (ULB, La Plaine).
To submit (in a *zip* file) a folder that contains at least following files, see Figure 1:

- all your C++ sources,
- a Makefile (that creates all executable files),
- A short report (pdf format, about 5 pages plus an appendix if needed, figures and graphics are welcome).

The zip file with your project has to be submitted to the UV: `uv.ulb.ac.be`. The name of the folder and of the zip file: if Chuck Norris makes his project with Michael Jackson they should send a file named *norris-jackson.zip* (that contains a folder of the same name, that contains all your project files).
If your project does not have a Makefile or if it does not compile or if it crashes during the execution or if it is missing the report or if it is not submitted using the described guidelines then your project will not be graded.
For questions on this project, please refer to **nikita.veshchikov@ulb.ac.be** with subject "INFO-F-404 project1".

Good luck!