# INFO-F404 - Real-Time Operating Systems
# Project 1: Global vs Partitioned DM

Anthony Caccia Jérôme Hellinckx

12 décembre 2016

## 1 Introduction

This project consists in studying performances of the *Deadline-monotonic scheduling* (DM) algorithm on systems with two different strategies : global and partitioned (best fit) scheduling. Systems have $n$ periodic, asynchronous and independant tasks $\tau$ with constrained deadlines on a multiprocessor system.

DM is a *Fixed Task Priority Scheduler* (FTP) : tasks are ranked deterministically by their relative deadlines. The lower the relative deadline, the higher the priority. In case of equality, the priority still has to be deterministic. Therefore, in this implementation, the the task id (which is unique) will decide of the priority.

### 1.1 Partitioned strategy

The main problem that has to be dealt with when using a partitioned strategy is to find an optimal partitioning such that each partition is schedulable on one processor while minimizing the number of partitions. Since this problem is known to be *NP-Complete*, this implementation uses the best-fit heuristic algorithm to find a partitioning. The *Best-Fit* strategy operates by assigning the current task to the processor for which the remaining utilization when adding said task utilization is minimum.

Once all tasks have been assigned to a partition, it cannot migrate and a uniprocessor scheduler is used on each partition.

### 1.2 Global strategy

Global strategy, on the other hand, permits to tasks to migrate between processors during their lifetime : they can then start their execution on a processor and resume on another one.

## 2 Code description

Three executables were asked in order to complete this project :

1. a simulator, taking as list of tasks, a number of processor and a strategy and simulate the system on the interval $I = [0; O_{max} + 2 * P]$ ;
2. a generator, which creates a list of $n$ tasks with an utilization $u$ ;
3. a study program, to test DM's performances.

### 2.1 Simulator

### 2.2 Generator

These are the primary constraints for tasks creation :

- there should be exactly $n$ tasks ;
- the system utilisation should be very close to $u$
- each generated tasks $\tau_i$ should have $0 <= u_i <= 1$
- $\forall \tau_i . t_i \geq d_i \geq c_i$

The code is really simple : you just generate $n$ numbers between 0 and 1, those numbers are then multiplied by the expected system utilisation $u$ and divided by their sum : those numbers are now the $u_i$ for each created tasks. We then convert $u_i$ to fraction : the numerator will become the $c_i$ and the denominator, the $t_i$. Now we can generate a $d_i$ which is a random number from an uniform integer distribution between $c_i$ and $t_i$.

All $o_i$ can be generated randomly, we will just substract each of them by the minimal $o_i$ so the minimal will be 0.

### 2.3 Study

## 3 Encoutered problems and solutions

### 3.1 Simulator

### 3.2 Generator

There could be problems in asked conditions : if the asked $u$ is bigger than $100 * n$, clearly it is impossible to generate $n$ tasks with $0 < u_i \leq 1$. If this case happen, $u$ is simply decreased to $100 * n$.

Contrarilly to what is stated in **??**, generating the first numbers between 0 and 1 is a bad idea. If we take an extreme case where it is asked to take $n$ jobs with a system utilisation $u = 100 * n$, it is logical that each task utilisation $\tau_u = 100$. We thus have to reduce the interval were the random numbers are taken. We can handle this by reducing this interval : we take $[m - s; m + s]$ where $m = u/n$ and $s = \min(m, 1 - m)$.

### 3.3 Study

## 4 Comparison tests specification

## 5 Results