INFO-F-404: Real-Time Operating Systems

Uniprocessor Scheduling Project

Joël Goossens **Yannick Molinghen**

1. Introduction

In this project, we consider synchronous task sets with constrained deadlines. This project consists in

- the implementation of three priority assignment algorithms (DM, EDF and Round Robin),
- a scheduler simulator,
- a report that compares the performance of DM, EDF and RR.

2. Project details

This project should be realised by group of two. There are two deliverables: the code itself preferably written in Rust or in Python[1] and a PDF report.

2.1. Task set file

Your program has to parse task set files. A task set file is a file that contains a description of a set of tasks with their respective offset O_i, computation time C_i, deadline D_i and period T_i. Task set files are simple Comma Separated Value (CSV) files where the line i encodes the O_i, C_i, D_i, T_i of task i. An example is shown here below where $\tau_1 = \{O_1 = 0, C_1 = 2, D_1 = 40, T_1 = 50\}$, and $\tau_2 = \{O_2 = 0, C_2 = 80, D_2 = 200, T_2 = 200\}$.

```
0, 20,  40,  50
0, 80, 200, 200
```

Note that the offset of each task is given in task set files even though it is systematically 0 in the scope of this project.

2.2. Scheduling algorithms

To determine the priority of each task, you have to implement Round Robin, Earliest Deadline First and Deadline Monotonic algorithms. Note that

- DM assigns priorities on the task level (FTP),
- EDF assigns priorities on the job level (FJP),
- Round Robin does not assign priorities per se.

2.3. Scheduler simulator

The purpose of the scheduler simulator is to determine if a task set is schedulable by a given algorithm.

Your scheduler should always take the fastest method possible to determine schedulability and only use simulation when it is necessary.

Simulating the execution of a task set involves computing a feasibility interval and then computing all of the events (job releases, deadline misses, job scheduling) that occur. Referring to your lecture notes, you have to determine what feasibility interval is applicable and use the smallest one[2]. You can choose whether you implement an event-based or a constant-step simulator.

[1]Contact me if you want to use another language

[2]You can use $[0, P)$ for Round Robin.

3. Running the program

3.1. Program inputs

The program should accept two mandatory arguments:

1. `dm, edf` or `rr`, the scheduling algorithm to use,
2. `<task set file>`, the path to the task set file.

You can add other **optional** arguments if you have any use for them.

```
$ python main.py audsley|edf|rr [-v] <taskset file> # python
$ cargo run audsley|edf|rr [-v] <taskset file>      # Rust
```

Parsing command line arguments can be tedious. Make your life easier and use a library such as `argparse` for Python or `clap` for Rust. It will result in a more readable code and a more robust parsing.

3.2. Process exit code

The exit code of the program will be used in automated tests to determine whether you properly detect which method to use to determine schedulability.

The program should exit with different values depending on how you defined the schedulability of the system. The exit codes and their meanings is shown in Table 1.

Exit code	**Description**
`0`	The task set is schedulable and you had to simulate the execution.
`1`	The task set is schedulable and you took a shortcut.
`2`	The task set is not schedulable and you had to simulate the execution.
`3`	The task set is not schedulable and you took a shortcut.

Table 1: Exit code for each case.

Important: To specify the exit code of your program, you can use `std::process::exit(<code>)` in Rust or `exit(<code>)` in Python. If your program does not exit with the appropriate code, it will not pass the automatic tests and your project will be poorly graded !

4. Experiments

You are provided with a dataset of synchronous task sets with constrained deadlines described in Section 4.1. You are required to analyse the dataset (Section 4.1) and compute the *success rate* of each scheduling algorithm on this dataset (Section 4.2).

We define the *success rate* of a task priority assignment algorithm A as the ratio of feasible task sets that A can schedule without missing a deadline. For instance, if 25% of the feasible task sets miss a deadline, then the success rate is 0.75. This is illustrated in below diagram representing the set of all possible task sets. The success rate is the ratio between the green area and the sum of the green and the red areas in Figure 1.

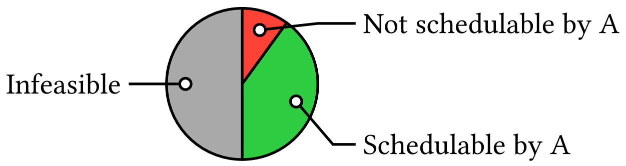

Figure 1: All possible task sets split in infeasible (grey) and feasible (others). Feasible task sets are themselves split into schedulable and not schedulable by a specific algorithm A.

4.1. Dataset

You can find in the "Université Virtuelle" a series of synchronous task sets with constrained deadlines grouped by utilisation and by number of tasks in the format discussed in Section 2.1. For instance, a task set with 20% utilisation and with 6 tasks will be in the directory `20-percent/6-tasks`.

For each utilisation and for each number of tasks, you have to determine the ratio of task sets that is feasible.

4.2. Plots

- Consider the task sets in the folder `80-percent/`.
  1. Plot the ratio of task sets that are feasible according to the number of tasks.
  2. Plot the success rate of each algorithm (RR, DM, EDF) according to the number of tasks.
- Consider the task sets in the folders `10-tasks/`.
  3. Plot the ratio of tasks that are feasible according to the utilisation.
  4. Plot the success rate of each algorithm (RR, DM, EDF) according to the utilisation.

Important: An inappropriate type of plot can be anything from unreadable to misleading. Take some time to think through which type of plot (pie, bar, line, scatter, Gantt, histogram…) is best suited for the job. Poorly chosen plot types will be penalised.

5. Report

Write a report that includes at least (and not especially in that order)

- a short introduction that defines the scope of the project
- your methodology
- your experimental setup
- the results of your experiments alongside with the value of the parameters
- a discussion of the results
- if applicable, what you have done in addition to the project statement.

6. Evaluation criteria

The project will be graded out of 20 points balanced as follows:

• Scheduling algorithms are correctly implemented	**/5**
• The code is readable, properly structured and uses appropriate abstractions	**/3**
• Report	**/12**
Total	**/20**

You have to hand in your project as a zip file containing your code and the report on the "Université Virtuelle" no later than the 10th of November 2024, 23:59.

Feel free to ask questions during practicals, via Teams or by email at yannick.molinghen@ulb.be.