UNIVERSITÉ LIBRE DE BRUXELLES ULB
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

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. You are required to implement four parts and write a report:

- Three priority assignment algorithms (DM, EDF and Round Robin)
- A task set generator.
- A scheduler to simulate the execution of a set of tasks
- A visualisation tool to visualise the execution of the scheduler

The main objective of the project is to compare the *success rate* of each algorithm (see Section 4).

2. Project components

This project should be realised by group of two. There are two deliverables: the code itself (preferably written in Python3 or in Rust, please contact me if you want to use another language) and a PDF report.

2.1. Task set file

Your program should be able 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 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: Since we only consider synchronous systems, the offset column could be inferred. However, this implementation aims at being future proof and hopefully used in the second project of this course where we will consider a wider range of task sets.

2.2. Priority assignment algorithms

A priority assignment algorithm has two main purposes:

- job priorisation: given a set of jobs and a time step, determine which job has the highest priority
- feasibility interval: given a task set, compute the smallest feasibility interval[1]

You should implement three priority assigment algorithms: Deadline Monotonic, Earliest Deadline First and Round Robin.

2.3. Scheduler simulator

The scheduler is the core of the project. It takes as input a task set and a scheduling algorithm and performs the scheduling simulation within the appropriate feasibility interval. The simulator is responsible for checking deadline misses.

[1] For Round Robin, you can use $[0, P)$ as feasibility interval.

2.4. Visualisation tool

You should implement a tool for the user to visualise the execution of a scheduling simulation, as shown in Figure 1. Deadline misses must be clearly indicated and job releases can be indicated.

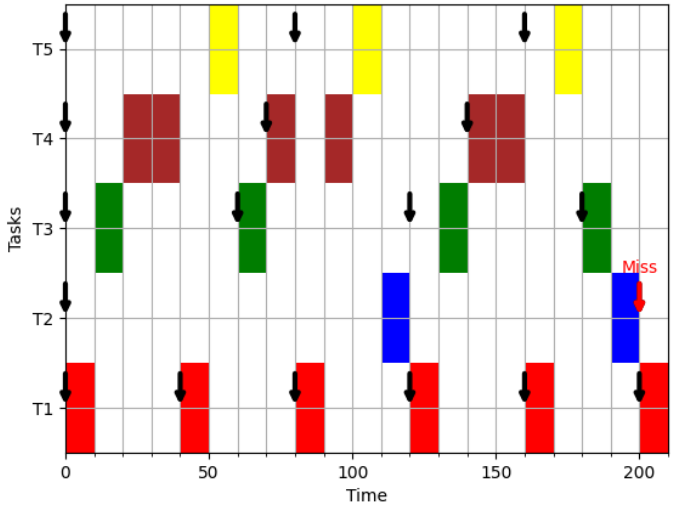

Figure 1: Example of schedule visualisation where a deadline is missed at t=200.

You can use libraries such as matplotlib for Python or plotters for Rust but feel free to use any other that is freely available if you prefer.

2.5. Task set generator

You have to implement a task set generator. It takes as input the number of tasks to generate and a target task set utilisation, and outputs a synchronous taskset with constrained deadlines. You will use this generator when performing the experiments described in Section 4.

3. Running the program

3.1. Program inputs

The program should accept two mandatory arguments, namely the algorithm to use (dm, edf or rr) and a task set file to read the tasks from. You can add other **optional** arguments if you have any use for them.

```
$ python main.py dm|edf|rr <taskset file> # python
$ cargo run dm|edf|rr <taskset file>      # Rust
```

Advice: To parse the command line arguments, make your life easier by using a library such as `argparse` if you program in Python, or `clap` in Rust.

3.2. Program outputs

At the end of the run, the program should save a figure that plots the schedule of the system as shown in Figure 1. The name of the file saved should be the same as the name of the input task set file with the name of the algorithm and a file extension (e.g. "taskset1" → "taskset1_edf.png"). Moreover, the exit code of your program should be 0 if the task set was schedulable, and 1 otherwise.

Important: If you do not exit with the appropriate return code, it will not pass the automatic tests that will be run on your project and will therefore be graded poorly. Make sure you exit the program with the correct exit code!

4. Experiments

We define the *success rate* of a task priority assignment algorithm A as the ratio of task sets that A can schedule without missing a deadline. For instance, if 25% of the task sets miss a deadline, then the success rate is 0.75.

In the following experiments, you have to compare the *success rate* of each algorithm, given a dataset of randomly generated task sets.

4.1. Running an experiment

1. Define the parameters of your experiments (random seed, time quantum in RR, number of tasks in a taskset, task set utilisation, ...)
2. Generate a dataset of at least 500 task sets with your generator
3. Run the scheduling simulation with each algorithm on this dataset
4. Compute the success rate of each algorithm

4.2. Plotting the results

You have to create two different plots. The first one shows the success rate of each algorithm according to the number of tasks (from 1 to 30). The second one shows the success rate of each algorithm according to the task set utilisation (from 0.05 to 1, every 0.05).

5. Report

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

- an introduction that defines the scope of the project
- the metodology
  - about the task generator
  - about the experiments
- the experimental setup
- the results of your experiments alongside with the value of the parameters
- a discussion of the results
  - results that were unexpected
  - conclusions that you can take from the results
  - discussion of how robust your results are
- 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 **/6**
- The code is readable, properly structured and uses appropriate abstractions **/4**
- Report
  - Conciseness, clarity and relevance of the content **/4**
  - Structure **/2**
  - Presentation and discussion of the results **/4**

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 12th of November 2023, 23:59.

Questions: Feel free to ask questions during practicals or by email at yannick.molinghen@ulb.be.