F404-RTOS Project 1 Report

Antoine Leyvraz: 498530, Yuwen Shen

November 8, 2024

1 Introduction

This project aims to evaluate the schedulability of synchronous task sets with constrained deadlines using Deadline Monotonic (DM), Earliest Deadline First (EDF), and Round Robin (RR) priority assignment on a uniprocessor system. We developed a Python program that takes as input the name of one of these priority assignment algorithms along with the task set information. The program automatically checks the feasibility and outputs the result.

2 Methodology

The methodology for checking schedulability is divided into two stages: preprocessing and simulation. In the preprocessing stage, the program assesses the feasibility of the input task set using shortcuts, such as utilization tests and response-time analysis. If preprocessing cannot determine feasibility and scheduling simulation is required, the preprocessor sets the feasibility interval and the simulation timestep. Then, a constant-step simulator is used to evaluate feasibility.

2.1 Data Types

To represent real concepts, we defined the classes `Task`, `Job`, and `Taskset` to support the program. The `Task` class can release a `Job` by creating a `Job` object based on the input time. Similarly, `Taskset` contains a list of `Task` objects; when `Taskset` releases a `Job`, each `Task` in the list releases a `Job` and adds it to a list of `Job` objects. The `Job` class includes methods to check if it has missed its deadline and to schedule itself by reducing its remaining computation time.

2.2 Preprocessing

The project focuses on synchronous task sets with constrained deadlines. Therefore, for DM and EDF, we can use shortcuts to determine schedulability, while for RR, simulation is always necessary. We encapsulated preprocessing functions within the `Preprocessor` class in the `src/preprocessor.py` file.

Upon receiving a task set, the program first verifies whether it is a synchronous, constrained-deadline system using the `check_taskset_properties()` function. Next, it checks feasibility with the `feasibility_check()` function. If the function indicates that simulation is necessary, the `set_feasibility_interval()` and `set_simulator_timestep()` functions are executed to configure the parameters for the simulation.

2.2.1 Check Task Set Properties

All tasks in the task set must have a zero offset (synchronous) and a deadline that is less than or equal to their period (constrained deadline). If each task's deadline is equal to its period, the program marks the task set as having an implicit deadline.

2.2.2 Check Feasibility (Shortcuts)

The program checks the system's feasibility in the following order. If a previous test determines the system's feasibility, the function returns and skips the remaining tests.

- **Utilization Test**: If the system utilization $U > 1$, the system is **infeasible**, and no simulation is needed.
- **Utilization Test for Implicit Deadlines with DM**: If the system is marked as having an implicit deadline, the DM algorithm defaults to Rate Monotonic (RM). The utilization test for RM is then applied. If $U \leq n(\sqrt[n]{2} - 1)$, where n represents the number of tasks, the system is **feasible** (Theorem 33), and no simulation is needed.
- **Deadline Monotonic (DM)**: If the scheduling algorithm is DM, a schedulability test based on the worst-case response time (wcrt) can determine feasibility without simulation (Theorem 29). The wcrt of each task is computed iteratively. If every task's wcrt is less than its deadline, the system is **feasible**; otherwise, it is **infeasible**.
- **Earliest Deadline First (EDF)**: If the system has implicit deadlines, it is EDF-**feasible** (Theorem 53), and no simulation is needed.
- **Simulation Required**: If none of the above tests determines system feasibility, a scheduling simulation is necessary.

2.2.3 Prepare Scheduling Simulation

If a scheduling simulation is necessary, the program prepares two key parameters before starting the simulation.

- **Feasibility Interval**:
  - **DM**: The feasibility interval for DM is the maximum deadline (Corollary 32).
  - **EDF**: The feasibility interval for EDF is initially set to the hyperperiod. However, if an idle point is found during the simulation, the system is considered EDF-feasible (Theorem 58), and the simulation will stop.

  - **RR**: The feasibility interval for RR is the hyperperiod.

- **Time Step**: The greatest common divisor of all tasks' computation time (C), deadline (D), and period (P).

2.3 Simulation

A constant-step simulator was developed to simulate the scheduling of the system. Using the parameters provided in Section 2.2.3, we defined scheduling functions for DM, EDF, and RR, which take a list of active `Job` objects and schedule the one with the highest priority according to the respective algorithm. If any job misses its deadline, the simulation stops, and the system is deemed infeasible with the selected algorithm. If the simulation reaches the end of the feasibility interval with no missed deadlines, the system is considered feasible with the selected algorithm. For the EDF algorithm, the simulation also stops when an idle point is found, as described in Section 2.2.3.

3 Experiments

3.1 Setup

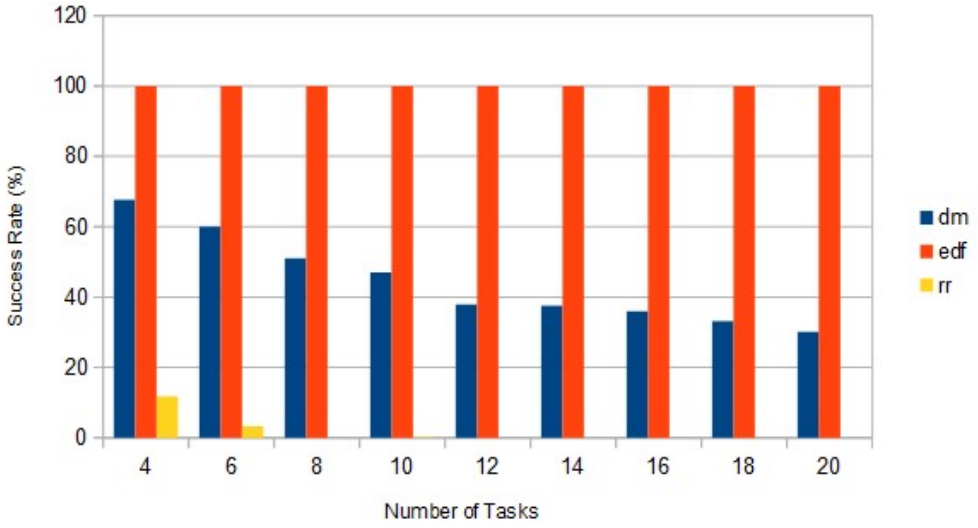
We used the original, more complex dataset consisting of two sections. The first section has a fixed utilization of 80% and a variable number of tasks, ranging from 4 to 20. The second section includes a fixed 10 tasks, but with utilization varying from 10% to 100%. We evaluated the feasibility of each task set using DM, EDF, and RR priority functions. This process is automated by the Python program `src/plot.py`.
Within this program, we would first run the preprocessor and simulator with EDF to check the feasibility of the task set, as any task set not schedulable by EDF should be infeasible. Then, if we were checking the task set on another algorithm such as DM or RR, we would run everything again with the chosen algorithm and filter all the returning runs according to their exit code to determine the amount of schedulable and unschedulable tasks within the given task set.
After checking all `.csv` task set files in a folder it uses the gathered data to generate a pie chart showing the percentage of task sets in the folder that are schedulable by the specified priority algorithm, unschedulable by the specified priority algorithm, or infeasible by any Fixed Job Priority (FJP) algorithms.

3.2 Results

Refer to appendices A and B at the end of the report.

3.3 Analysis

3.3.1 80 Percent task sets

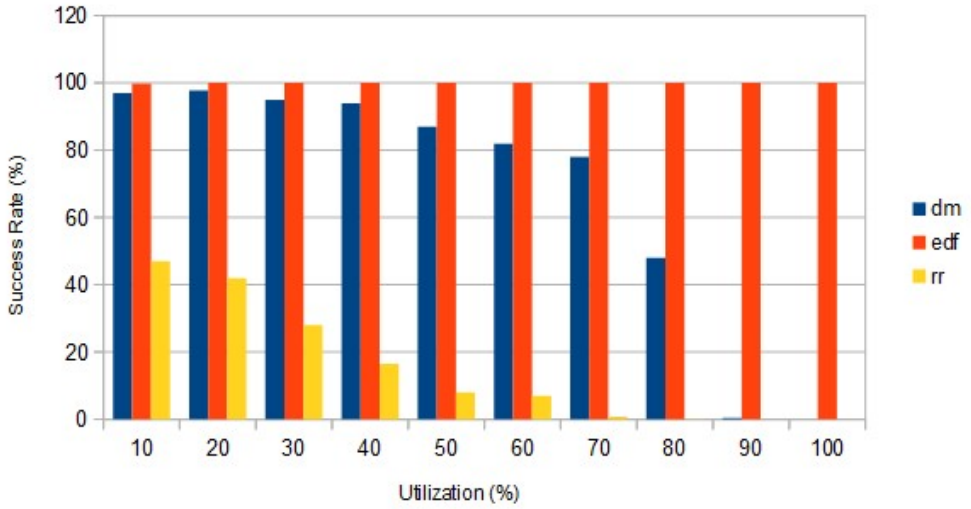


Evolution of success rate by the algorithm over the number of tasks on 80-Percent

From the data in Appendix A, it is clear that "Earliest Deadline First" (EDF) is by far the most effective scheduling algorithm of the three. This result aligns with expectations, as EDF is optimal among all Fixed Job Priority (FJP) scheduling algorithms. Any task set deemed FJP-feasible is guaranteed to be schedulable by EDF. Assuming there is no context switch cost, the number of tasks within the set does not affect the schedulability of the EDF algorithm.
"Deadline Monotonic" (DM), on the other hand, performs relatively well with a small number of tasks, showing a success rate of about 68% with 4 tasks. However, as the number of tasks increases, its efficiency decreases somewhat linearly, achieving a success rate of approximately 30% with 20 tasks. This decline is understandable, as the system utilization, when randomly distributed across a larger number of tasks, increases the likelihood of high-utilization tasks with short deadlines being assigned higher priority in DM. This can cause lower-priority tasks to miss their deadlines.
"Round Robin" (RR) is by far the weakest performer of the group. At 80% utilization, even with only 4 tasks, RR struggles to achieve more than approximately a 12% success rate. This success rate drops quickly, reaching a flat 0% from 8 to 20 tasks. Since RR evenly distributes computation power among tasks without considering each task's utilization and deadline, an increase in the number of tasks raises the likelihood of having high-utilization

tasks with relatively shorter deadlines, leading to deadline misses in the RR algorithm.

3.3.2 10 Tasks task sets



Evolution of success rate by the algorithm over the utilization on 10-Tasks

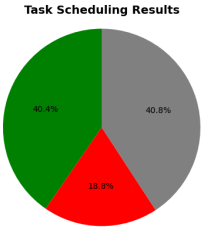
If we take a look at appendix B, we can have a fairer picture of the algorithms performance depending on the total utilization of the task set. RR for example achieves a close to 48% success rate with 10 tasks at 10% utilization. Though you can definitely observe a sharp decline as utilization rises. It succeeds only about 4% of the time at 70% utilization and is at 0% for anything above that.
EDF is still at the top on this part of the data set, still achieving a 100% success rate on every challenge, because it is FJP optimal. DM on the other hand, whilst achieving a respectable 97% at 10% utilization. It still holds up respectably up to 70% utilization as it succeeds 78% of the time but, above that, we can observe a sharp decline with around 48% at 80% utilization and falling to basically 0% above that.
In conclusion, we confirm the optimality of the EDF algorithm among FJP scheduling algorithms. Any FTP-feasible system is also FJP-feasible, and therefore EDF-feasible. However, checking feasibility for EDF requires simulation, which can be costly for systems with constrained deadlines. DM is also a strong candidate, especially for systems with low to average utilization. It can likely schedule many tasks efficiently at low utilization levels. Since its feasibility can be verified with the worst-case response time (wcrt), simulation is unnecessary for DM, reducing computational time. RR, on the other hand, has proven to be nearly ineffective in most scenarios. It performs decently at very low utilization, but while it is the fastest of the three algorithms, its simplicity ultimately limits its effectiveness.

3.4 Appendices

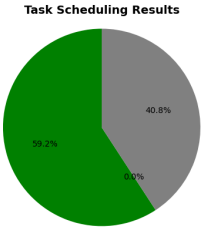
For the whole dataset the chosen color code was the following:
- Green → Schedulable by the algorithm
- Red → Unschedulable by the algorithm
- Gray → FJP infeasible

3.4.1 Appendix A: 80 Percent

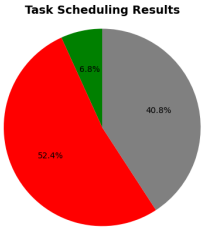
4 Tasks



DM

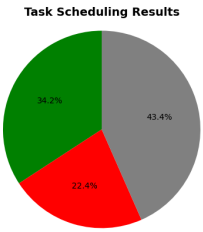


EDF

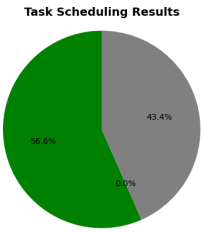


RR

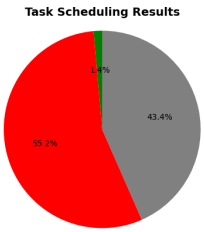
6 Tasks



DM

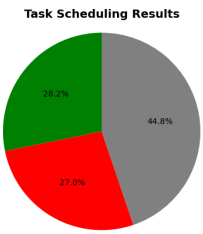


EDF

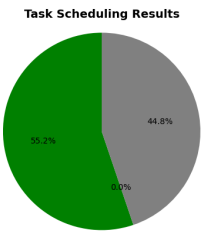


RR

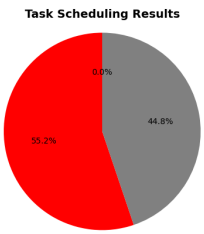
8 Tasks



DM

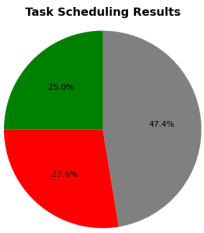


EDF

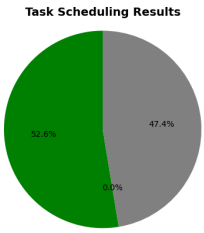


RR

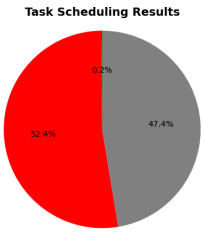
10 Tasks



DM

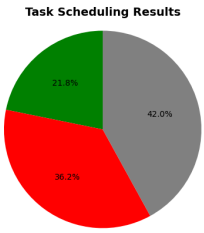


EDF

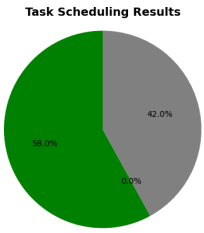


RR

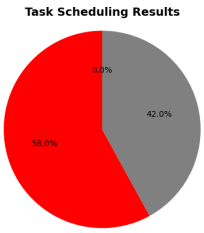
12 Tasks



DM

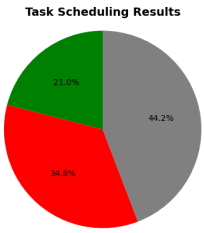


EDF

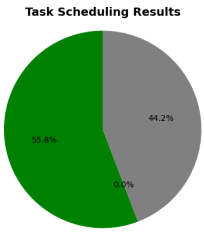


RR

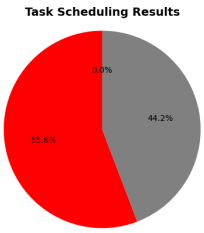
14 Tasks



DM

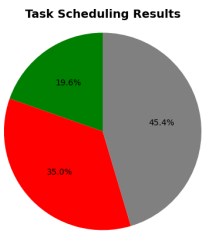


EDF

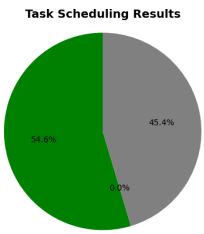


RR

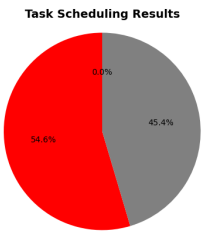
16 Tasks



DM

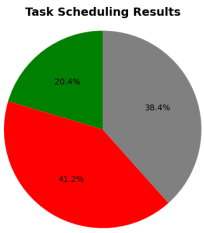


EDF

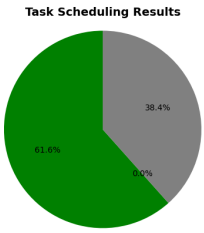


RR

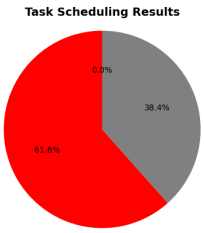
18 Tasks



DM

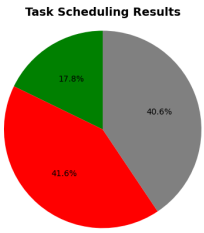


EDF

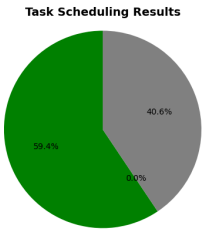


RR

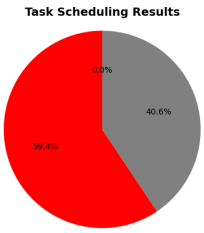
20 Tasks



DM



EDF



RR

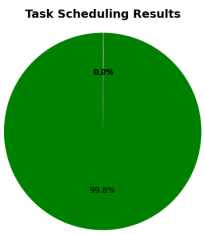
	dm	edf	rr
4	67.7%	100%	11.8%
6	60%	100%	3.3%
8	51%	100%	0%
10	47%	100%	0.2%
12	37.9%	100%	0%
14	37.5%	100%	0%
16	35.9%	100%	0%
18	33.1%	100%	0%
20	30.1%	100%	0%

Table 1: Success rate for each algorithm for each amount of tasks

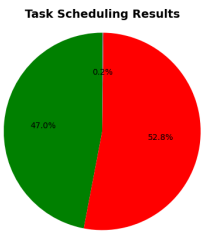
3.4.2 Appendix B: 10 Tasks

10 Percent

DM

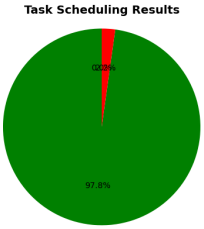


EDF

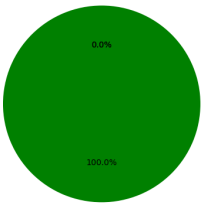


RR

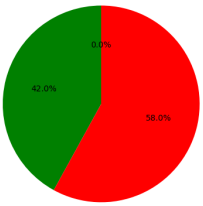
20 Percent



DM

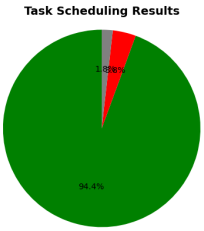


EDF

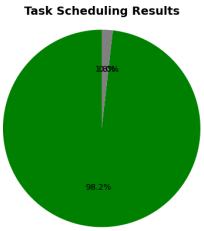


RR

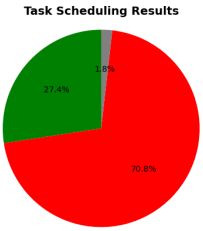
30 Percent



DM

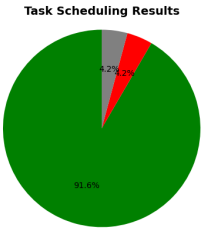


EDF

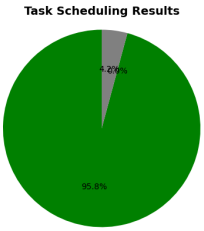


RR

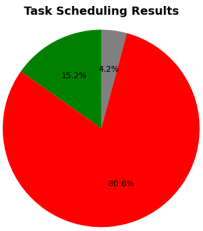
40 Percent



DM

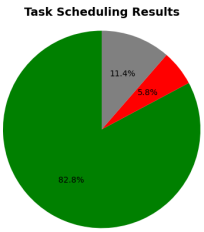


EDF

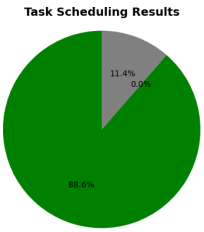


RR

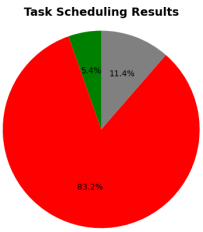
50 Percent



DM

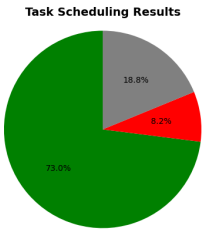


EDF

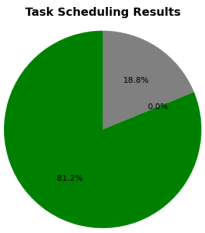


RR

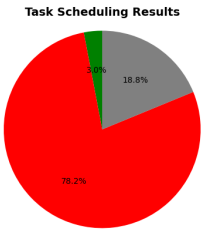
60 Percent



DM

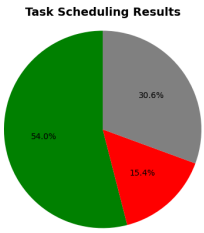


EDF

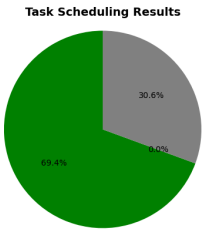


RR

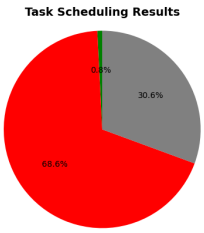
70 Percent



DM

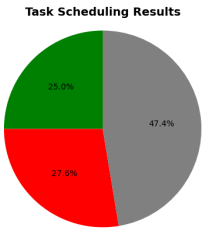


EDF

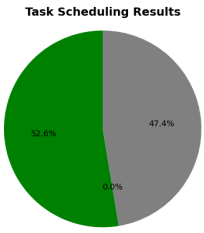


RR

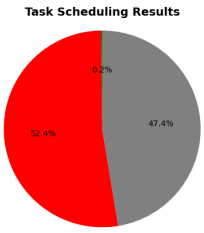
80 Percent



DM

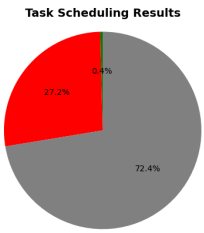


EDF

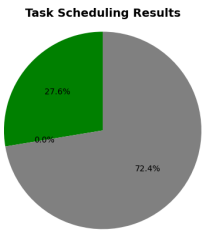


RR

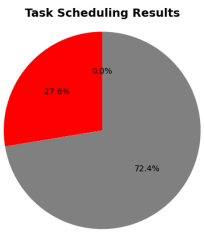
90 Percent



DM

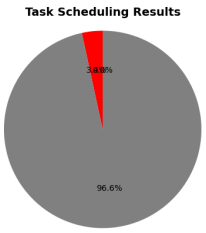


EDF

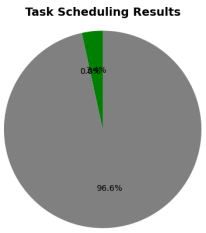


RR

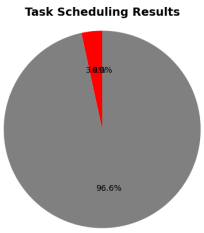
100 Percent



DM



EDF



RR

	dm	**edf**	**rr**
10	97%	100%	47%
20	97.8%	100%	42%
30	95%	100%	28%
40	94%	100%	16%
50	87%	100%	8%
60	82%	100%	7%
70	78%	100%	0.8%
80	48%	100%	0.2%
90	0.7%	100%	0%
100	0%	100%	0%

Table 2: Success rate for each algorithm for each utilization percentage