ÉCOLE D'INGÉNIEURS
DE L'AIR, DE L'ESPACE ET DE LA MOBILITÉ DURABLE

FINAL ASSIGNMENT : Schedule search

Élèves :
- APARICIO GIMENO Pablo

Niveau :
- A4

Année universitaire 2024-2025
M. SINGH Jasdeep

Introduction :

This report analyzes the schedulability of a periodic task set using job-level scheduling implemented in Python. Each task is defined by its computation time and period. The goal is to verify whether all tasks can meet their deadlines and, if so, to compute a schedule that minimizes total waiting time and maximizes processor utilization. Additionally, a second scenario is considered where task τ_5 is allowed to miss its deadline to further reduce waiting time. Key assumptions, schedulability analysis, and algorithm complexity. The task set is composed of seven periodic tasks, each with a computation time (C) and a period (T). You can find the project code :

https://github.com/PabloAPAGIM/Temps-reel

1 Task Set

The task set is composed of seven periodic tasks, each with a computation time (C) and a period (T). The task parameters are summarized in Table **??** :

Task	C	T_i
τ_1	2	10
τ_2	3	10
τ_3	2	20
τ_4	2	20
τ_5	2	40
τ_6	2	40
τ_7	3	80

TABLE 1 – Task set

2 Schedulability of the Task Set

To determine schedulability by utilization, we use the formula :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} < 1$$

For the given task set :

$$U = \frac{2}{10}+\frac{3}{10}+\frac{2}{20}+\frac{2}{20}+\frac{2}{40}+\frac{2}{40}+\frac{3}{80} = 0.2+0.3+0.1+0.1+0.05+0.05+0.0375 = 0.8375 < 1$$

Since the total utilization is less than 1, and we are performing exact response time analysis below, the task set is schedulable under preemptive scheduling. Then we can calculate the Hyperperiod of our system composed of tasks with periods $T_1 = 10$, $T_2 = 10$, $T_3 = 20$, $T_4 = 20$, $T_5 = 40$, $T_6 = 40$, and $T_7 = 80$, the hyperperiod is calculated as :

$$\text{Hyperperiod} = \text{LCM}(10, 20, 40, 80) = 80$$

So we can confirm the schedulability of our task system.

2.1 Assumptions

We make the following assumptions :

— **Single-core Processor :** The system is assumed to be equipped with a single-core processor. This means that at any given instant, only one task can be executed.

— **Independent Tasks :** All tasks are considered independent. There is no resource sharing, no mutual exclusion, and no use of critical sections. As a result, there are no blocking times to consider during response time analysis.

3 Python Code for Schedule Generation

Now I will present the Python code I developed to find a valid schedule. I was not able to implement a code that could check every possible schedule permutation. Thus, my code uses a heuristic : at each time unit, it chooses the available task whose absolute deadline is the earliest. This allows us to guarantee that no deadlines are missed. However, it is possible that the total execution time and the total waiting time could be further optimized with a different combination of task executions.

The Python script generates a chart representing the execution timeline of the tasks during one hyperperiod (0 to 80 time units). Each task is displayed with a unique color, and periods of processor idleness are shown in black.

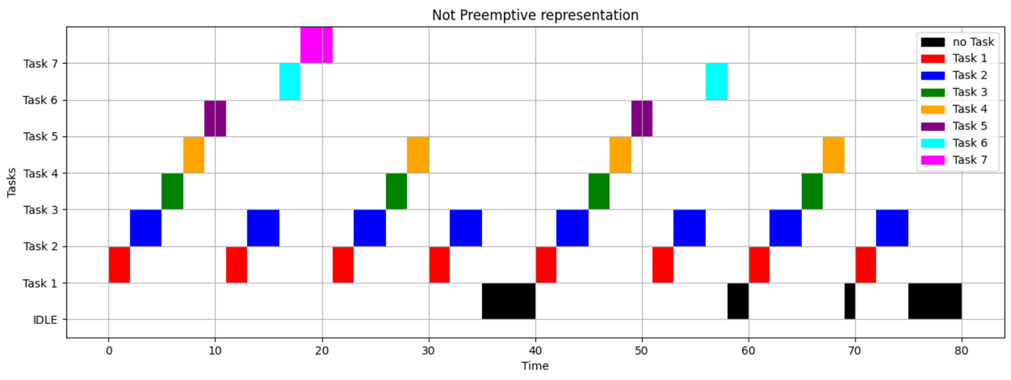


FIGURE 1 – Chart non Preemptive

3.1 Calculation of Waiting Time and Response Time

In real-time systems, for each job J_{ij} :

— The **Response Time** (R_{ij}) is defined as the time elapsed between the release of the job and its completion.

$$R_{ij} = \text{Finish Time} - \text{Release Time}$$

It can also be recursively computed using response-time analysis formulas provided in the lecture :

$$R_{ij} = R_{ij}^{p_{ij-1}} + C_{ij}$$

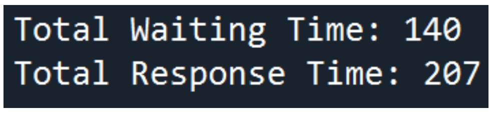
where C_{ij} is the computation time and $R_{ij}^{p_{ij-1}}$ represents interference from higher priority tasks.

— The **Waiting Time** (W_{ij}) is defined as :

$$W_{ij} = R_{ij} - C_{ij}$$

It measures the amount of time the job spends waiting in the ready queue before starting its execution.

The total waiting time and total response time across all jobs was computed after the schedule simulation, getting :



```
Total Waiting Time: 140
Total Response Time: 207
```

FIGURE 2 – Total waiting and response time

3.2 Algorithmic Complexity

The developed heuristic algorithm has a much lower complexity than an exhaustive search. Here are the different cases :

— **Exhaustive search :** To check all possible job execution orders would require evaluating all permutations, leading to a factorial complexity :

$$O(hyperperiod * n!) \quad \text{with} \quad n = \text{number of jobs}$$

For our case, even considering only 7 tasks, an exhaustive search would involve $7! = 5040$ combinations, which is impractical.

— **Current heuristic (Earliest Deadline First selection) :** At each time step, the algorithm scans all available jobs to select the one with the earliest deadline. Thus, the complexity per time step is $O(n)$, and over the entire hyperperiod, it becomes :

$$O(hyperperiod \times n)$$

where n is the number of tasks.

4 Conclusion

The task set was found to be schedulable using a Python implementation based on an Earliest Deadline First heuristic. All deadlines were respected, and total waiting and response times were calculated. Although the heuristic does not guarantee the absolute minimum waiting time, it provides an efficient and practical schedule without exhaustive search. Allowing τ_5 to miss its deadline was identified as a possible way to further reduce waiting time, but was not fully implemented.