Institut Polytechnique des Sciences Avancées

Reel Time Embedded System :

Final assignement : Schedule Search

FOUCHARD Axel

University Year 2024-2025

Table des matières

I Introduction

In this project, we want to study the **schedulability** of an set of real time tasks in a **non préemptive way**. The objective was to :
— Verify the schedulability of the system,
— Generate a schedule ensuring all tasks meet their deadlines,
— Minimize the overall waiting time,
— Allow a certain task (τ_5) to miss her deadline.

II Tasks

The tasks to planify is :

Task	**C**	**T**
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

III Test of schedulability

We applied the following schedulability condition :

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

So we compute :

$$\text{Global Utilization} = \frac{2}{10} + \frac{3}{10} + \frac{2}{20} + \frac{2}{20} + \frac{2}{40} + \frac{2}{40} + \frac{3}{80} \approx 0.838$$

Conclusion : the system is **schedulable**.

IV Methodology

1 Computation of the Hyperperiod

We determined the hyperperiod as the lowest common multiple of the periods T :

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

2 Job Generation

Each task τ_i generates a new job every T_i units of time, until the hyperperiod. Each job is defined by :
— Its release time,
— Its absolute deadline,
— Its computation time C_i.

3 EDF Non-Preemptive Algorithm

The EDF algorithm tends to minimize the waiting time by prioritizing jobs with the earliest deadline. This is why I decided to choose this scheduler for next steps.

— At every time step, the job with the closest deadline among the ready jobs is selected.
— Once a job starts executing, it runs until completion without interruption.
— In case of ties, the job from the task with the lowest index is chosen.

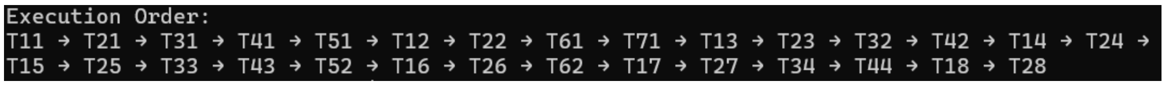
4 Task τ_5 Allowed to Miss Deadlines

In a second simulation scenario, task τ_5 was allowed to miss its deadlines if necessary. This enabled the system to prioritize more urgent jobs during periods of high load.

V Results

1 Execution Order

Here is how the different jobs are scheduled :

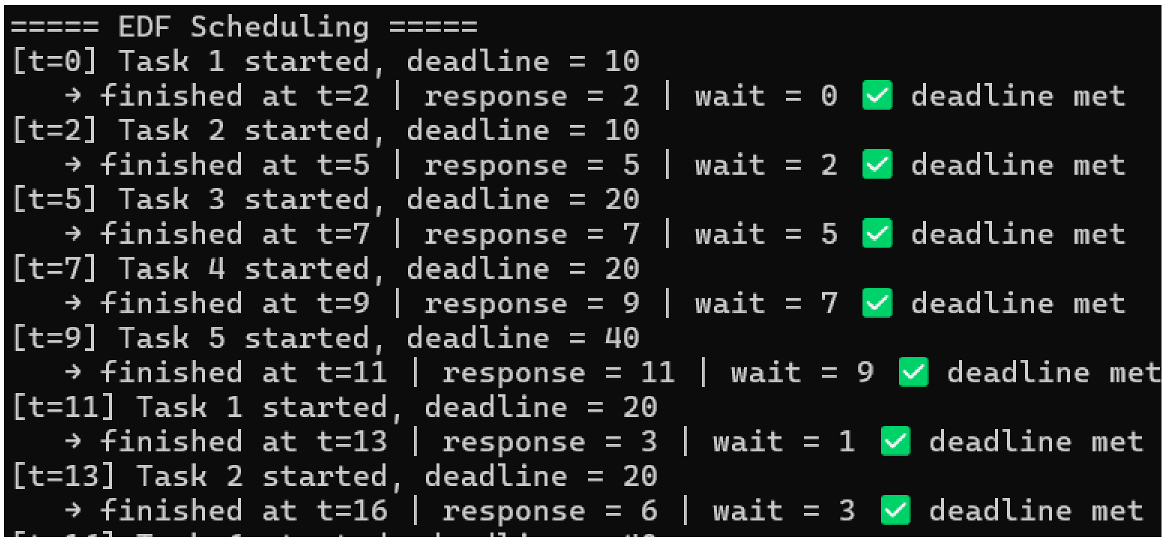


```
Execution Order:
T11 → T21 → T31 → T41 → T51 → T12 → T22 → T61 → T71 → T13 → T23 → T32 → T42 → T14 → T24 →
T15 → T25 → T33 → T43 → T52 → T16 → T26 → T62 → T17 → T27 → T34 → T44 → T18 → T28
```

FIGURE 1 – Schedule of the Tasks

Each identifier represents the j-th job of task i (e.g., T21 is the first job of task 2).

2 Deadline Compliance



```
===== EDF Scheduling =====
[t=0] Task 1 started, deadline = 10
   → finished at t=2 | response = 2 | wait = 0 ✅ deadline met
[t=2] Task 2 started, deadline = 10
   → finished at t=5 | response = 5 | wait = 2 ✅ deadline met
[t=5] Task 3 started, deadline = 20
   → finished at t=7 | response = 7 | wait = 5 ✅ deadline met
[t=7] Task 4 started, deadline = 20
   → finished at t=9 | response = 9 | wait = 7 ✅ deadline met
[t=9] Task 5 started, deadline = 40
   → finished at t=11 | response = 11 | wait = 9 ✅ deadline met
[t=11] Task 1 started, deadline = 20
   → finished at t=13 | response = 3 | wait = 1 ✅ deadline met
[t=13] Task 2 started, deadline = 20
   → finished at t=16 | response = 6 | wait = 3 ✅ deadline met
```

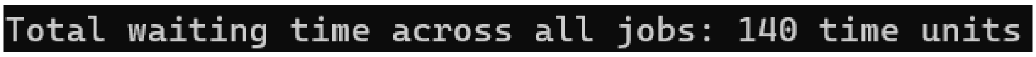
FIGURE 2 – EDF Process

In order to follow the different steps, we show for each jobs : the starting and ending time, the number of the task, the response time, the waiting time before the execution of the job compare to when it could have started and if the deadline is met or not.

All tasks met their deadlines **even for** τ_5 in the second scenario, as allowed.

3 Waiting Time Analysis

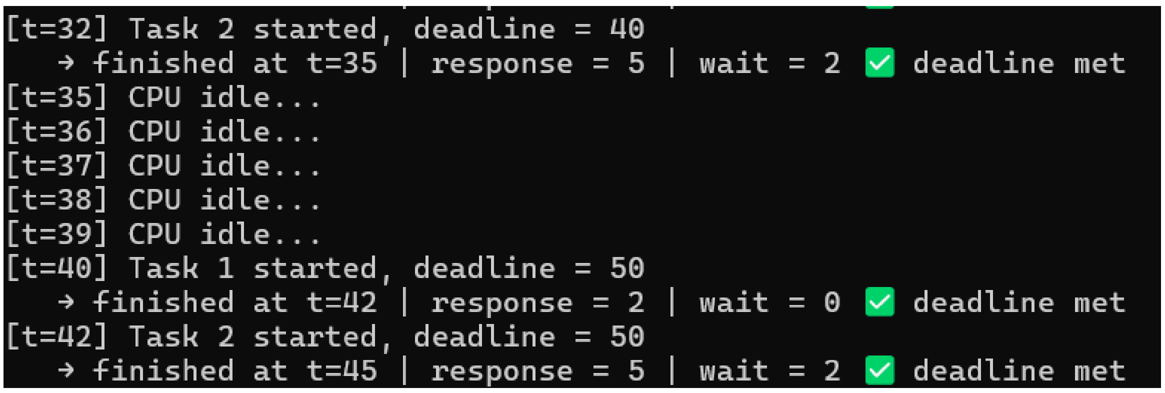
EDF scheduling naturally tends to minimize waiting time, as it prioritizes jobs with the most urgent deadlines. While not guaranteed to yield the absolute minimum waiting time (especially in non-preemptive contexts), it remains highly effective in practice.



```
Total waiting time across all jobs: 140 time units
```

FIGURE 3 – Total Waiting time

As we can see in the **Figure 4**, there is as well some moment where none of the tasks are running.



```
[t=32] Task 2 started, deadline = 40
   → finished at t=35 | response = 5 | wait = 2 ✅ deadline met
[t=35] CPU idle...
[t=36] CPU idle...
[t=37] CPU idle...
[t=38] CPU idle...
[t=39] CPU idle...
[t=40] Task 1 started, deadline = 50
   → finished at t=42 | response = 2 | wait = 0 ✅ deadline met
[t=42] Task 2 started, deadline = 50
   → finished at t=45 | response = 5 | wait = 2 ✅ deadline met
```

FIGURE 4 – No task running

VI Computational Complexity

The computational complexity of the EDF scheduler implementation is analyzed in the following components :

— **Job generation :** For each task, a number of jobs is generated proportional to the hyperperiod divided by the task's period. Assuming n tasks and H as the hyperperiod, the number of generated jobs is approximately $\sum_{i=1}^{n} H/T_i$. This results in a complexity of $O(n \cdot H)$ in the worst case.

— **Job selection (EDF) :** At each time unit, the scheduler scans through the list of active jobs to select the one with the earliest deadline. If m is the number of total jobs generated over the hyperperiod, then job selection at each time unit has complexity $O(m)$. Over the full hyperperiod, this yields a complexity of $O(H \cdot m)$.

— **Execution and monitoring :** Each job, once selected, is executed without preemption. For each executed job, metrics such as response time and waiting time are recorded, contributing an additional linear cost in the number of jobs : $O(m)$.

Total complexity : Considering that m is proportional to $n \cdot H$, the overall complexity of the simulation can be approximated by :

$$O(n \cdot H) + O(H \cdot n \cdot H) + O(n \cdot H) = O(n \cdot H^2)$$

This complexity is acceptable for small to moderately sized task sets but may become computationally expensive for large hyperperiods or systems with many tasks.

VII Conclusion

We have demonstrated that :

— The task set is schedulable under EDF,

— The EDF non-preemptive algorithm ensures deadline compliance and manages load effectively,

— Allowing τ_5 to miss deadlines provides better feasibility under heavy load.

The scheduling logic as been implemented in C, and validated through simulation. All jobs were scheduled within the computed hyperperiod, and execution order, response time and deadline were logged.