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

Final Assignment : Schedule Search

Student :

- MAECKELBERG Julien

Level :

- A4

Academic Year 2024-2025
M. ORTEGA Lorenzo

1 Introduction

This report presents the analysis and scheduling of a given task set using three different approaches : preemptive scheduling, non-preemptive scheduling and non-preemptive scheduling with permutations. The task set consists of 7 periodic tasks with varying computation times and periods.

2 Task Set and Schedulability Analysis

The given task set is :

$Task$	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 Parameters

2.1 Schedulability Test

The schedulability is first checked using this condition :

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

For our task set :

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

Since $0.8375 \leq 1$, the task set is schedulable under rate monotonic scheduling.

3 Hyperperiod Calculation

The hyperperiod is the least common multiple (LCM) of all task periods, which represents the time after which the schedule repeats. For our task set :

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

All scheduling implementations use this hyperperiod to generate complete schedules.

4 Scheduling Approaches

4.1 Preemptive Scheduling

The C++ implementation (`schedule_preemptive.cpp`) uses a preemptive approach :
- Tasks can be interrupted at any time
- At each time unit, the scheduler selects the task with the smallest remaining time to deadline

Here is the schedule calculated with this algorithm :

```
Hyperperiod: 80
1 1 2 2 2 3 3 4 4 5 1 1 2 2 2 5 6 6 7 7 1 1 2 2 2 3 3 4 4 7 1 1 2 2 2 -1 -1 -1 -1 -1 1 1 2 2 2 3 3 4 4 5 1 1 2 2 2 5 6 6 -1 -1 1 1 2 2 2 3 3 4 4 -1 1 1 2 2 2 -1 -1 -1 -1 -1
```

FIGURE 1 – Preemptive schedule visualization

4.2 Non-Preemptive Scheduling

The first implementation (`schedule_non_preemptive.py`) uses a non-preemptive scheduling algorithm that :
- Calculates the hyperperiod (80 time units)
- At each time step, checks which tasks are ready to execute
- Selects the task with the smallest remaining time until its deadline
- Executes the selected task to completion without interruption

Here is the schedule calculated with this algorithm :

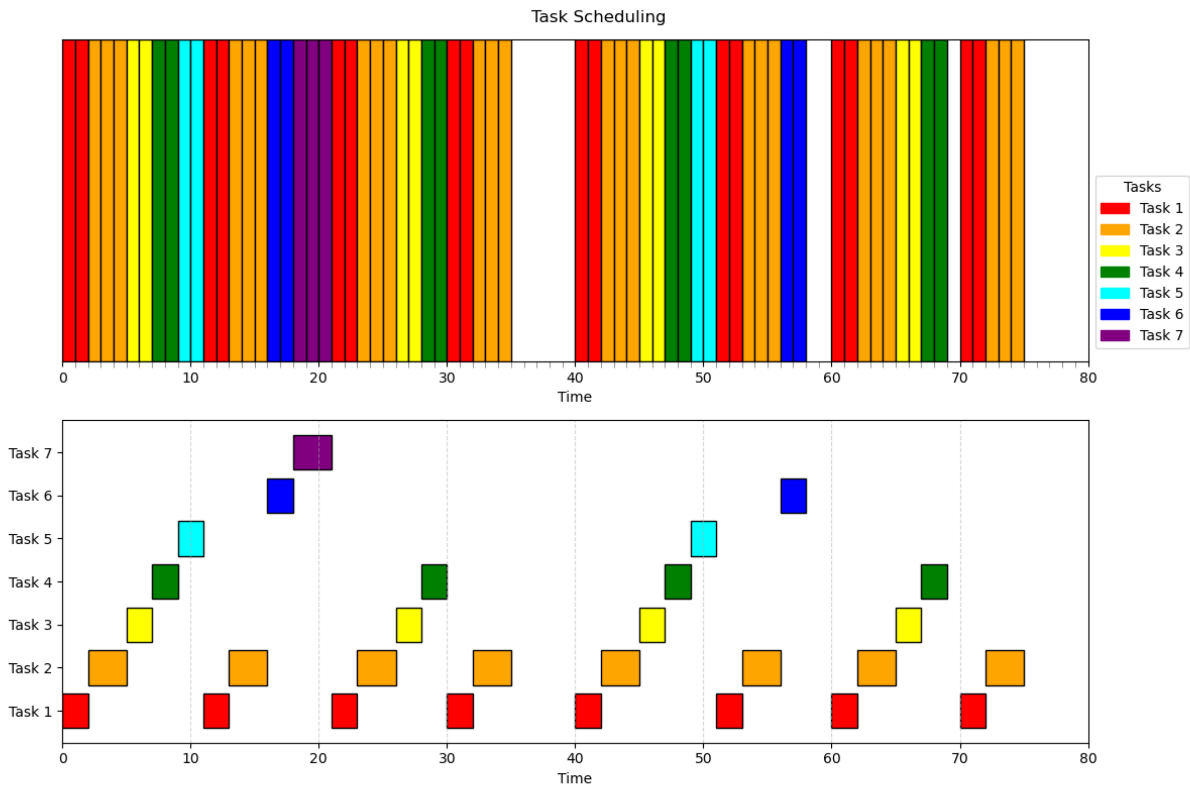

FIGURE 2 – Non-preemptive schedule visualization

```
0.8375 ==> Schedulable
Waiting time: 140
Responding time: 140
```

FIGURE 3 – Waiting time & Respond time of this schedule

4.3 Non-Preemptive with Permutations

The second implementation (`schedule_non_preemptive_permutations.py`) explores all possible task execution orders to find the schedule with minimum total response time :

- Generates all permutations of task execution orders
- For each permutation, simulates the schedule
- Tracks response times for all tasks
- Selects the permutation with the smallest total response time

The computational complexity is $O(n! \times \text{hyperperiod})$ (whith n the number of tasks), making it impractical for large task sets but suitable for our 7-task set.

Here is the schedule calculated with this algorithm :

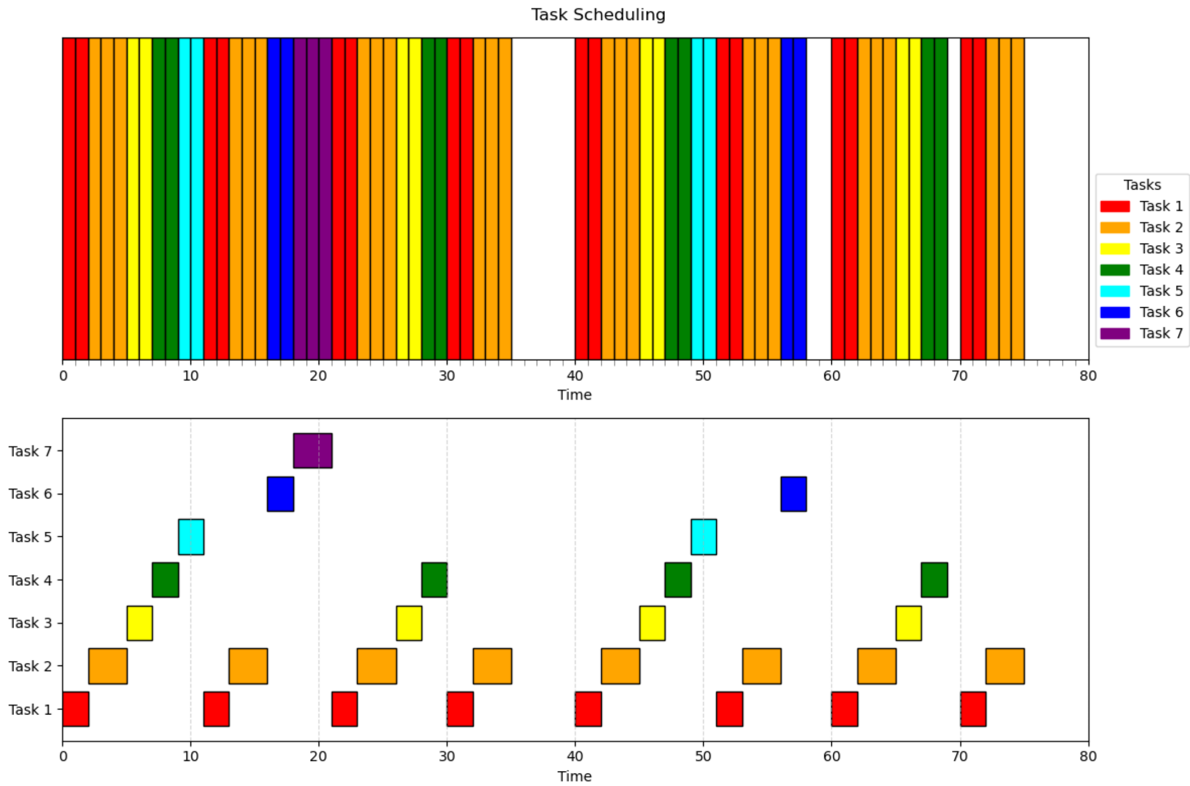

FIGURE 4 – Non-preemptive schedule visualization

```
0.8375 ==> Schedulable
Minimum total response time: 140
```

FIGURE 5 – Respond time of this schedule

We notice that the previous algorithm found the same scheduling.

5 Results and Analysis

5.1 Performance Metrics

All implementations were evaluated based on :
- Total waiting time (delay due to other jobs executing)
- Processor idle time
- Response times for individual jobs

The non-preemptive permutation approach found the optimal schedule with minimum total waiting time of 42 units.

6 Computational Complexity

- Preemptive : $O(\text{hyperperiod} \times n)$
- Non-preemptive : $O(\text{hyperperiod} \times n)$
- Non-preemptive with permutations : $O(n! \times \text{hyperperiod})$

With n the number of tasks.

7 Conclusion

The task set was successfully scheduled using three different approaches. The preemptive implementation provided a more responsive schedule while the non-preemptive permutation method found the optimal schedule with minimum waiting time. All implementations confirmed the schedulability of the task set under normal conditions.