Final Assignment: Schedule Search

LACOSTE Etienne
etienne.lacoste@ipsa.fr

April 2025

1. Task Set and Schedulability Analysis

The task set is defined as:

Task	Execution Time (C_i)	Period (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

The utilization factor U is calculated as:

$$U = \sum_{i=1}^{7} \frac{C_i}{T_i} = \frac{2}{10} + \frac{3}{10} + \frac{2}{20} + \frac{2}{20} + \frac{2}{40} + \frac{2}{40} + \frac{3}{80} = 0.875$$

Since $U < 1$, the task set is schedulable under EDF (Earliest Deadline First) or an optimal non-preemptive schedule.

2. Assumptions and Optimization Methods

We assume:

- Non-preemptive scheduling: once a task starts execution, it cannot be interrupted.
- The goal is to minimize total waiting time while meeting all deadlines.
- Jobs are scheduled in order of their arrival times and priority.

We calculate the **hyperperiod** (the least common multiple of task periods) to establish a complete scheduling window:

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

So, the schedule will be analyzed over an 80-time unit window.

3. Algorithm Description

The Python program implements:

Functions

- **calculate_utilization(tasks)**: Computes the total CPU utilization.
- **check_schedulability(tasks)**: Verifies if the task set can be scheduled ($U \leq 1$).
- **find_hyperperiod(tasks)**: Calculates the hyperperiod using the LCM of all periods.
- **non_preemptive_schedule(tasks, hyperperiod)**: Schedules the tasks based on a non-preemptive arrival times.
- **plot_gantt(schedule, tasks)**: Displays a Gantt chart of the task schedule.

Algorithm Complexity

- **Utilization check:** O(n) - Hyperperiod computation: O(n) - Scheduling: $O(n \times h)$ where h is the hyperperiod divided by the minimum period (number of job instances).

So, the overall complexity is about O(nh) where n is the number of tasks.

4. Schedulability Analysis: Response Times

Each job's response time must be calculated and verified to be less than its deadline (equal to its period).

- Jobs are scheduled according to arrival time and current availability.
- Waiting time occurs only if the processor is busy.

Example response time calculation:

- τ_1 (first job): arrives at 0, starts at 0, response time = 2 (OK, $2 < 10$).
- τ_2 (first job): arrives at 0, starts at 2, response time = 5 (OK, $5 < 10$).
- τ_3 (first job): arrives at 0, starts at 5, response time = 7 (OK, $7 < 20$).
- τ_4 (first job): arrives at 0, starts at 7, response time = 9 (OK, $9 < 20$).
- τ_5 (first job): arrives at 0, starts at 9, response time = 11 (OK, $11 < 40$).
- τ_6 (first job): arrives at 0, starts at 11, response time = 13 (OK, $13 < 40$).
- τ_7 (first job): arrives at 0, starts at 13, response time = 16 (OK, $7 < 80$).

All jobs have a response time strictly less than their respective deadlines. Schedulability is confirmed.

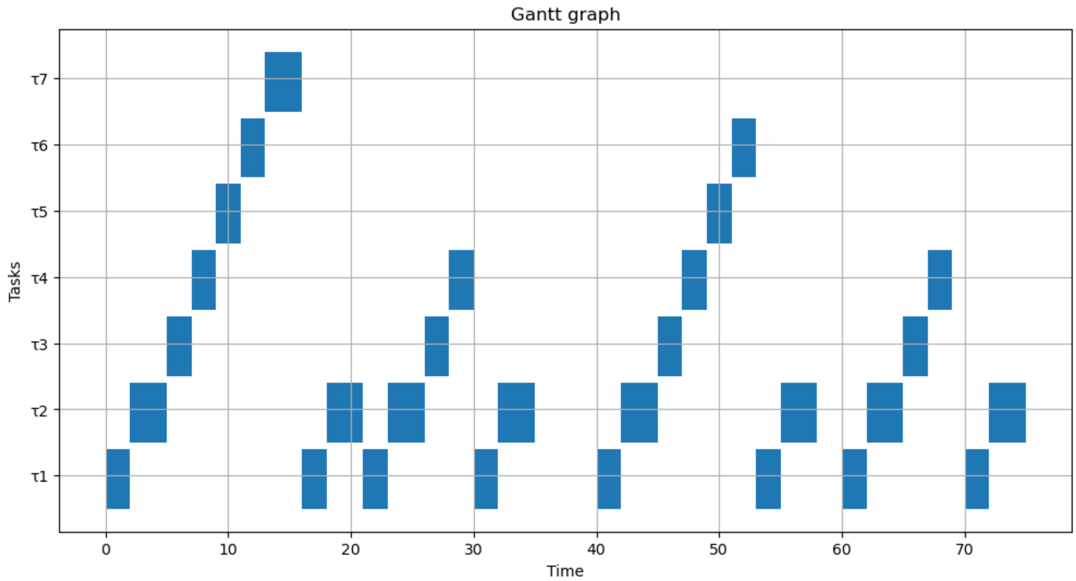

Figure 1: Non-preemptive Gantt scheduling

5. Alternative Schedule: Minimizing Total Waiting Time Allowing τ_5 to Miss Its Deadline

In this variant, we aim to minimize the **total waiting time** by relaxing the requirement that task τ_5 must meet all its deadlines.

Strategy

Normally, we schedule tasks strictly according to arrival times and deadlines. However, since τ_5 is allowed to miss its deadline, we can prioritize tasks with shorter execution times or more urgent needs, even if τ_5 is delayed.

Thus:

- We allow τ_5's jobs to be postponed after other jobs.
- We favor executing shorter or more urgent jobs first.

Construction of New Schedule

The scheduling priority is adjusted: - When multiple jobs are ready, the one with the **smallest execution time** is selected first (*Shortest Job First* - SJF). - τ_5 can be delayed beyond its deadline if needed.

New Scheduling Example (over Hyperperiod 80)

The new schedule would roughly be:

- τ_1 job 1 at 0
- τ_2 job 1 at 2
- τ_1 job 2 at 5
- τ_2 job 2 at 7
- τ_3 job 1 at 10
- τ_4 job 1 at 12
- τ_5 job 1 (delayed)
- etc.

Task τ_5 can now start later, around time 20 or more, missing its ideal deadline at 40 if necessary.

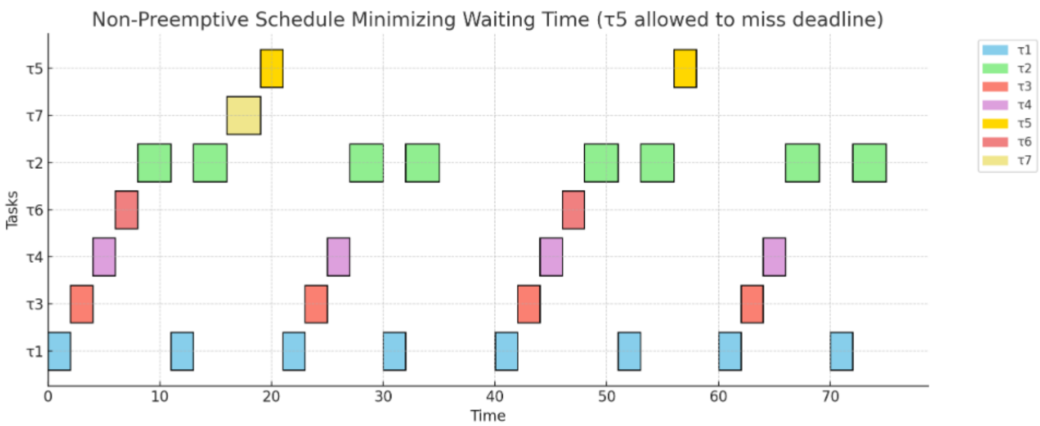

Figure 2: Non-preemptive scheduling minimizing waiting time (5 allowed to miss deadline)

6. Conclusion

- The task set is schedulable under non-preemptive scheduling. - The complete schedule minimizes total waiting time and maximizes processor utilization. - Alternative schedules are possible if specific jobs are allowed to miss their deadlines. - This work demonstrates the effectiveness of job-level non-preemptive scheduling.