Question

- If the density of a system is larger than 1, the system **must be infeasible**??

  How about $T_1 = (2, 0.9, 1)$, $T_2 = (5, 2.3)$?
  (Δ = 0.9 + 0.46 = 1.36 > 1)

- **Theorem 2.** A system $\mathbf{T}$ of independent, preemptable tasks can be feasibly scheduled on one processor if its density is equal to or less than 1.

Schedulability Test (1/2)

- We call a test for the purpose of validating that the given application system can indeed meet all its hard deadlines when scheduled according to the chosen scheduling algorithm a schedulability test.

Schedulability Test (2/2)

- We are given
  - the period p_i, execution time e_i, and relative deadline D_i of every task T_i in a system $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ of independent periodic tasks, and
  - a priority-driven algorithm used to schedule the tasks in $\mathbf{T}$ preemptively on one processor.

  We are asked to determine whether all the deadlines of every task T_i, for every $1 \leq i \leq n$, all always met.

Schedulability Test for the EDF (1/2)

- From **Theorem 1** & **2**, to determine whether the given system of n independent periodic tasks surely meets all the deadlines when scheduled according to the preemptive EDF algorithm on one processor, we check whether the inequality

$$\sum_{k=1}^{n} \frac{e_k}{\min(D_k, p_k)} \leq 1$$

  is satisfied. We call this inequality the schedulability condition of the EDF algorithm.

Schedulability Test for the EDF (2/2)

- If $D_k \geqq p_k$ for all k from 1 to n, the equation is both a necessary and sufficient condition for a system to be feasible.
- If $D_k < p_k$ for some k, the equation is only a sufficient condition; therefore we can only say that the system may not be schedulable when the condition is not satisfied (from Theorem 2).

We can use the schedulability condition of the EDF algorithm as a rule to guide the choices of the periods and execution times of the tasks while we design the system.

Example

- Consider a digital robot controller
  - Control-law computation:
    - Takes no more than 8ms to complete
    - Executes once every 10ms
  - Built-In Self-Test (BIST):
    - The maximum execution time: 50ms
    - We can execute the BIST task as frequently as once every 250ms
  - Telemetry task:
    - Must execute 15ms

If we are willing to reduce the frequency of the BIST task to once a second, we can make the relative deadline of the telemetry task as short as 100ms.

Outline

- Assumptions
- Fixed-Priority vs. Dynamic-Priority Algorithms
- Maximum Schedulable Utilization
- *Optimality of the RM and DM Algorithms*
- A Schedulability Test for Fixed-Priority Tasks with Short Response Times
- Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times
- Sufficient Schedulability Conditions for the RM and DM Algorithms

Simplification

- Hereafter in our discussion on fixed-priority scheduling, we index the tasks in decreasing order of their priority except where stated otherwise. In other words, the task T_i has a higher priority than the task T_k if $i < k$.
- We refer to the priority of a task T_i as priority π_i, π_i*'s* are positive integers $1, 2, \ldots, n$, 1 being the highest priority and n being the lowest priority.
- We denote the subset of tasks with equal or higher priority than T_i by $\mathbf{T}_i$ and its total utilization by $U_i = \sum_{k=1}^{i} u_k$.

Limitation of Fixed-Priority Algorithms

- Fixed-priority algorithms cannot be optimal: Such an algorithm may fail to schedule some systems for which there are feasible schedules.
- **Example:** $T_1 = (2, 1)$ and $T_2 = (5, 2.5)$
  - The tasks are feasible (from **Theorem 1**).
  - In the time interval $(0, 4]$, T_1 must have a higher-priority than T_2.
  - At time 4, T_2 must have a higher-priority than T_1.
- While the RM algorithm is not optimal for tasks with arbitrary periods, it is optimal in a special case.

Simply Periodic

- A system of periodic task is simply periodic if for every pair of tasks T_i and T_k in the system and $p_i < p_k$, p_k is an integer multiple of p_i.

- **Theorem 3.** A system of simply periodic, independent, preemptable tasks whose relative deadlines are equal to or larger than their periods is schedulable on one processor according to the RM algorithm if and only if its total utilization is equal to or less than 1.

Informal Proof of Theorem 3

- Assumptions:
  - Tasks are in phase (i.e., the tasks have identical phases).
  - The processor never idles before the task T_i misses a deadline for the first time at t, where t is an integer multiple of p_i.
- The total time required to complete all the jobs with deadlines before and at t is

$$\sum_{k=1}^{i}(e_k t / p_k) = t\sum_{k=1}^{i} u_k = tU_i$$

- That T_i misses a deadline at t means that this demand for time exceeds t, i.e., $U_i > 1$.

Advantages of Fixed-Priority Algorithms

- Despite the fact that fixed-priority scheduling is not optimal in general, we may nevertheless choose to use this approach because it leads to a more predictable and stable system.
- **Theorem 4.** A system $\mathbf{T}$ of independent, preemptable periodic tasks that are in phase and have relative deadlines equal to or less than their respective periods can be feasibly scheduled on one processor according to the DM algorithm whenever it can be feasibly scheduled according to any fixed-priority algorithm.

Why Theorem 4 is True?

- Because we can always transform a feasible fixed-priority schedule that is not a DM schedule into one that is.
  1. Sort all tasks with relative deadlines.
  2. Switch tasks T_i and T_{i+1}, which do not follow the DM rule. (No deadline will be missed, why?)
  3. Repeat Step 2. until all tasks are prioritized according to the DM rule.
- **Corollary 1.** The RM algorithm is optimal among all fixed-priority algorithms whenever the relative deadlines of the tasks are proportional to their periods.

Outline

- Assumptions
- Fixed-Priority vs. Dynamic-Priority Algorithms
- Maximum Schedulable Utilization
- Optimality of the RM and DM Algorithms
- *A Schedulability Test for Fixed-Priority Tasks with Short Response Times*
- Schedulability Test for Fixed-Priority Tasks with Arbitrary Response Times
- Sufficient Schedulability Conditions for the RM and DM Algorithms

Pseudo-polynomial Time Schedulability Test

- We confine our attention to the case where response times of the jobs are smaller than or equal to their respective periods.
- Every job completes before next job on the same task is released.
- **References**

  - J. Lehoczky, L. Sha, and Y. Ding, “The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior,” in IEEE Real-Time System Symposium, 1989.
  - Audsley, N. Burns, A. Richardson, M. Tindell, K. Wellings, A. J., “Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling,” Software Engineering Journal, 1993, pp.284-292.

Outline

- A Schedulability Test for Fixed-Priority Tasks with Short Response Times:
  1. ***Critical Instants***
  2. Time-Demand Analysis
  3. Alternatives to Time-Demand Analysis

Critical Instants (1/2)

- The schedulability test checks one task T_i at a time to determine whether the response times of all its jobs are equal to or less than its relative deadline D_i.
- Because we cannot count on any relationship among the release times to hold, we must first identify the worst-case combination of release times of any job $J_{i,c}$ in T_i and all the jobs that have higher priorities than $J_{i,c}$.

Critical Instants (2/2)

- A critical instant of a task T_i is a time instant which is such that
  - If the *response time of every job in* T_i *is equal to or less than the relative deadline* D_i *of* T_i

    The job in T_i released at the instant has the maximum response time of all jobs in T_i.
  - If the *response time of some jobs in* T_i *exceeds* D_i

    The response time of the job released at the instant is greater than D_i.

Maximum (Possible) Response Time

- We call the response time of a job in T_i released at a critical instant the maximum (possible) response time of the task and denote it by W_i.
- **Theorem 5.** In a fixed-priority system where every job completes before the next job in the same task is released, a critical instant of any task T_i occurs when one of its job $J_{i,c}$ is released at the same time with a job in every higher-priority task, that is, $r_{i,c} = r_{k,l_k}$ for some l_k for every $k = 1, 2, \ldots, i - 1$.

  Proof. Please see the handout.