Topic I: Priority-Driven Scheduling of Periodic Tasks

謝仁偉 助理教授
jenwei@mail.ntust.edu.tw
國立台灣科技大學 資訊工程系
2008 Fall

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
- Practical Factors

Assumptions

- The tasks are independent.
  - When tasks are independent, the scheduler can delete any task and add an acceptance task at any time without causing any missed deadline.
  1. Introduce other resources and discuss the effects of resource contention.
  2. Describe resource access-control protocols designed to keep bounded the delay in job completion caused by resource contentions.
- No aperiodic and sporadic tasks
  - Integrate the scheduling of aperiodic and sporadic tasks with periodic tasks.

Task Model Assumptions

- Every job is ready for execution as soon as it is released.
- Every job can be preempted at any time.
- Every job never suspend itself.
- Scheduling decisions are made immediately upon job releases and completions.
- The context switch overhead is negligibly small compared with execution times of the tasks.
- The number of priority levels is unlimited.

➡ We will remove these restrictions and discuss the effects of these and other practical factors!

Terminologies

- We refer to periodic tasks simply as tasks.
- We use the term period to mean *the minimum inter-release time* of jobs in a task.

Let's Start…

1. An application creates a new task.
2. The application requests the scheduler to add the new task by providing the scheduler with relevant parameters of the task, including its period, execution time, and relative deadline.
3. Based on these parameters, the scheduler does an acceptance test on the new periodic task:
   - It accepts and adds the new task to the system only if the new task and all other existing tasks can be feasibly scheduled.
   - Otherwise, the scheduler rejects the new task.

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
- Practical Factors

Priority-Driven Scheduler

- A priority-driven scheduler is an on-line scheduler which assigns priorities to jobs after they are released and places the jobs in a ready job queue in priority order according to some priority-driven algorithm.
- When preemption is allowed at any time, a scheduling decision is made whenever a job is released or completed.

Two Types of Priority-Driven Algorithms (1/2)

- A fixed-priority algorithm assigns the same priority to all the jobs in each task.
  - The priority of each periodic task is fixed relative to other tasks.
  1. Rate-Monotonic Algorithm
  2. Deadline-Monotonic Algorithm

Two Types of Priority-Driven Algorithms (2/2)

- A dynamic-priority algorithm assigns different priorities to the individual jobs in each task.
  - The priority of the task with respect to that of other tasks changes as jobs are released and completed.
  1. Task-level dynamic-priority (and job-level fixed-priority) algorithms, e.g., EDF algorithm.
  2. Job-level (and task-level) dynamic-priority algorithms, e.g., LST algorithm.

Outline

- Fixed-Priority vs. Dynamic-Priority Algorithms:
  1. ***Rate-Monotonic and Deadline-Monotonic Algorithms***
  2. Well-Known Dynamic Algorithms
  3. Relative Merits

Rate-Monotonic Algorithm

- Assign priorities to tasks based on their period: the shorter the period, the higher the priority.
- The rate (of job releases) of a task is the inverse of its period.
- We will refer to this algorithm as the RM algorithm for short and a schedule produced by the algorithm as an RM schedule.
- Example of a RM Schedule:
  $T_1 = (4, 1), T_2 = (5, 2), T_3 = (20, 5)$

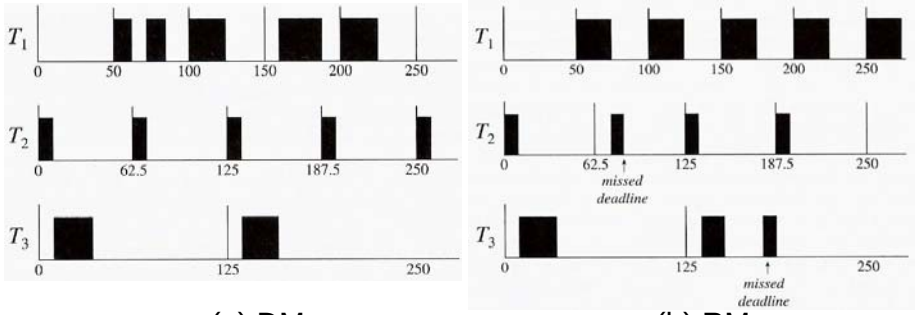
Deadline-Monotonic Algorithm

- Assign priorities to tasks according their relative deadlines: the shorter the relative deadline, the higher the priority.
- Notation: (*phase*, *period*, *execution time*, *deadline*)

Fixed-Priority Schedules

- T_1 = (50, 50, 25, 100), T_2 = (0, 62.5, 10, 20), T_3 = (0, 125, 25, 50)

(phase, period, execution time, deadline)



(a) DM

$\tau_2 > \tau_3 > \tau_1$

(b) RM

$\tau_1 > \tau_2 > \tau_3$

RM vs. DM

- When the relative deadline of every task is proportional to its period, the RM and DM algorithms are identical.
- When the relative deadlines are arbitrary, the DM algorithm performs better in the sense that it can sometimes produce a feasible schedule when the RM algorithm fails, while the RM algorithm always fails when the DM algorithm fails.

Outline

- Fixed-Priority vs. Dynamic-Priority Algorithms:
  1. Rate-Monotonic and Deadline-Monotonic Algorithms
  2. ***Well-Known Dynamic Algorithms***
  3. Relative Merits

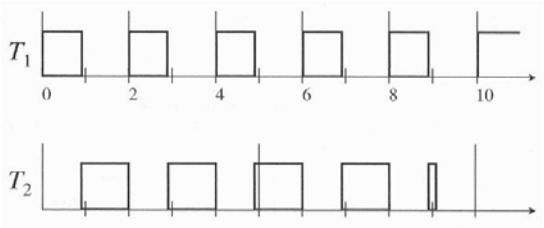
Earliest Deadline First Algorithm

- The Earliest-Deadline-First (EDF) algorithm assigns priorities to individual jobs in the tasks according to their absolute deadlines.

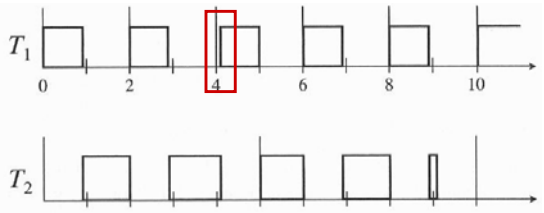
EDF Schedule vs. RM Schedule

- $T_1 = (2, 0.9)$, $T_2 = (5, 2.3)$
- EDF algorithm is a task-level dynamic-priority algorithm, but a job-level fixed-priority algorithm.

RM Schedule



EDF Schedule



Least-Slack-Time First Algorithm

- Least-Slack-Time First (LST) algorithm
  - At time t, the slack of a job whose remaining execution time is x and whose deadline is d is equal to $d - t - x$.
  - The scheduler checks the slacks of all the ready jobs each time a new job is released and orders the new job and the existing jobs on the basis of their slacks: the smaller the slack, the higher the priority.
  - LST algorithm is a job-level dynamic-priority algorithm.

Nonstrict LST vs. Strict LST

- Nonstrick LST: scheduling decisions are made only when jobs are released or completed.
- Strict LST: reassigns priorities to jobs whenever their slacks change relative to each other.
- The run-time overhead of the strict LST algorithm includes the time required to monitor and compare the slacks of all ready jobs as time progresses.
- By letting jobs with equal slacks execute in a round-robin manner, these jobs suffer extra context switches.

Outline

- Fixed-Priority vs. Dynamic-Priority Algorithms:
  1. Rate-Monotonic and Deadline-Monotonic Algorithms
  2. Well-Known Dynamic Algorithms
  3. ***Relative Merits***

Performance Criterion

- A criterion we use to measure the performance of algorithms used to schedule periodic tasks is the schedulable utilization.
- Schedulable Utilization of the Algorithm: A scheduling algorithm can feasibly schedule any set of periodic tasks on a processor if the total utilization of the tasks is equal to or less than the schedulable utilization of the algorithm.
- Since no algorithm can feasibly schedule a set of tasks with a total utilization greater than 1, an algorithm whose schedule utilization is equal to 1 is an optimal algorithm.

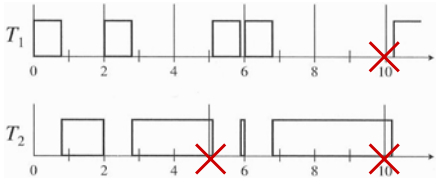
Advantage of Fixed-Priority Algorithms

- Although optimal dynamic-priority algorithms outperform fixed-priority algorithms, an advantage of fixed-priority algorithms is predictability.

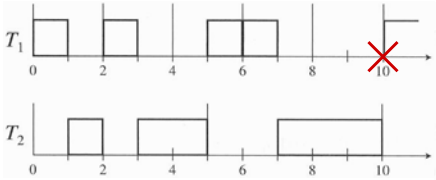
➡ When tasks have fixed priorities, overruns of jobs in a task can never affect higher-priority tasks!

Unpredictability and Instability of the EDF Algorithm

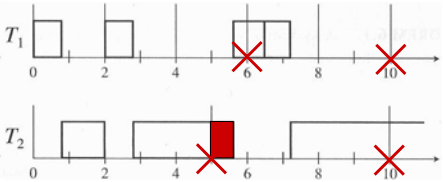
- There is no easy test, short of an exhaustive one, that allows us to determine which tasks will miss their deadlines and which tasks will not.



(b) $T_1 = (2, 0.8)$, $T_2 = (5, 3.5)$; $U = 1.1$



(a) $T_1 = (2, 1)$, $T_2 = (5, 3)$; $U = 1.1$



(c) $T_1 = (2, 0.8)$, $T_2 = (5, 4.0)$; $U = 1.2$

Disadvantages of the EDF Algorithm

- Unpredictable during an overload.
- If the execution of a late job is allowed to continue, it may cause some other jobs to be late.

➡ The scheduler should either lower the priorities of some or all the late jobs, or discards some jobs if they cannot complete by their deadlines and logs this action.

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
- Practical Factors

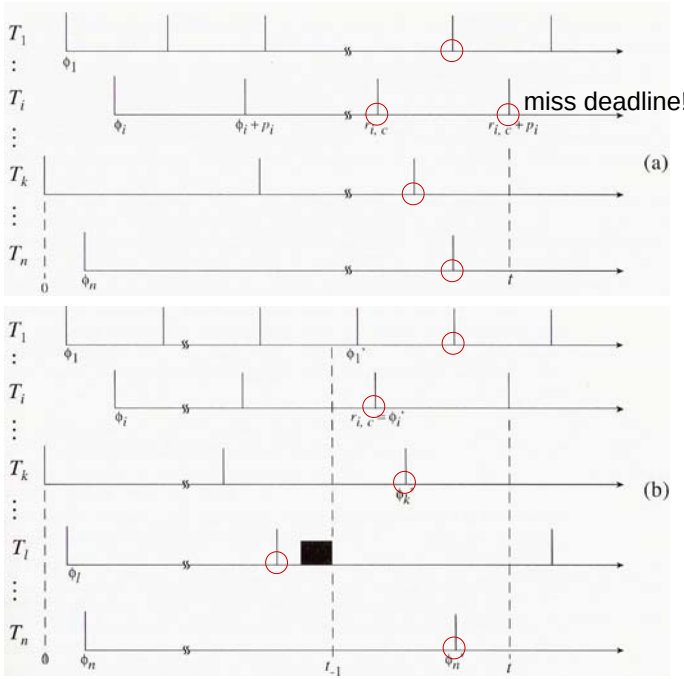
Definition

- Suppose a task set is scheduled by *S* scheduling algorithm, the schedule for the task set is feasible if all the jobs in each task can meet their corresponding deadlines under *S* scheduling. A task set is schedulable if there exists a feasible schedule.
- At any time t, the current period of a task is the period that begins before t and ends at or after t.
- We call the job that is released in the beginning of the current period the current job.

Schedulable Utilizations of the EDF

- **Theorem 1.** A system T of independent, preemptable tasks *with relative deadlines equal to their respective periods* can be feasibly scheduled on one processor if and only if its total utilization is equal to or less than 1.

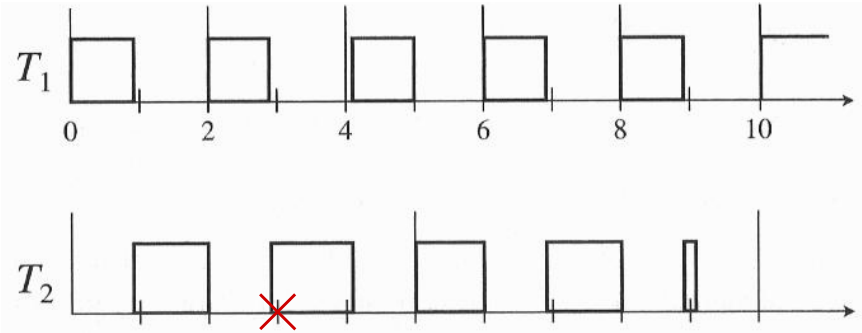
  Proof. Please see the handout.



From Theorem 1, We Know That…

- A system of independent, preemptable periodic tasks with *relative deadlines longer than their periods* can be feasibly scheduled on a processor as long as the total utilization is equal to or less than 1.
- The schedulable utilization $U_{EDF}(n)$ of the EDF algorithm for n independent, preemptable periodic tasks with relative deadlines equal to or larger than their periods is equal to 1.

How about deadline less than period?



- $T_1 = (2, 0.9)$, $T_2 = (5, 2.3, 3)$

Density of the System

- We call the ratio of the execution time e_k of a task T_k to the minimum of its relative deadline D_k and period p_k the density of the task. In other word, the density of T_k is $e_k / \min(D_k, p_k)$.
- The sum of the densities of all tasks in a system is the density of the system and is denoted by Δ.
- When $D_i < p_i$ for some task T_i, $\Delta > U$.

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 $1 \leqq i \leqq n$, 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

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, …, 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
- Practical Factors

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*

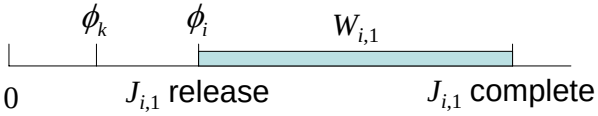
    The job in T_i released at the instant has the maximum response time of all jobs in T_i.
  - If the *response time of some jobs in T_i exceeds D_i*

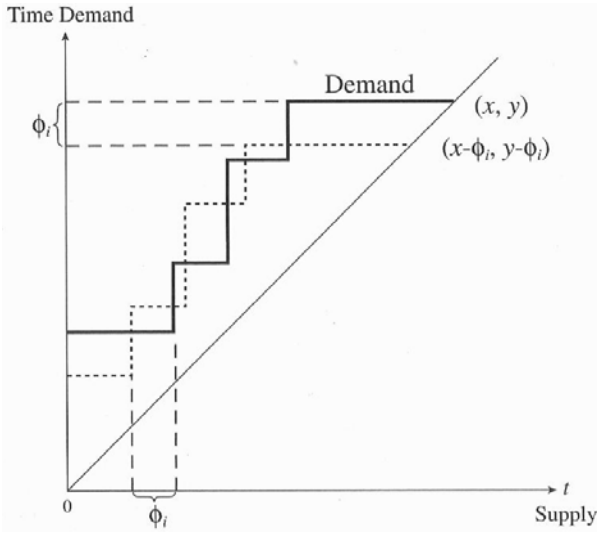
    The response time of the job released at the instant is greater than D_i.

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 $k = 1, 2, \ldots, i-1$.

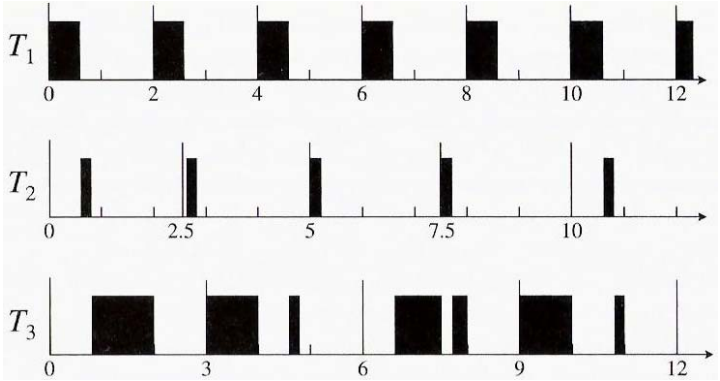
  Proof. Please see the handout.





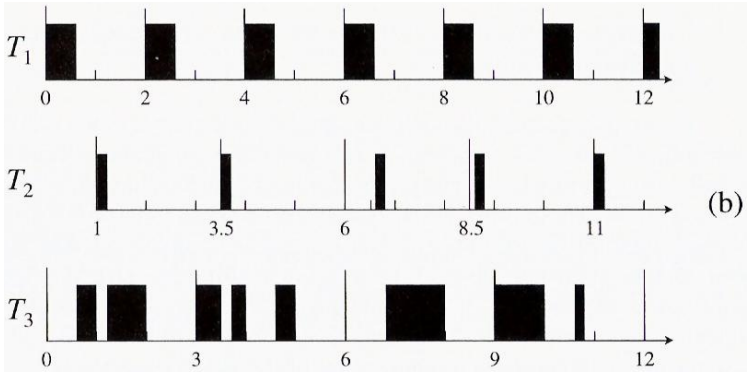
Example of Critical Instants (1/2)

- $T_1 = (2, 0.6)$, $T_2 = (2.5, 0.2)$, $T_3 = (3, 1.2)$
  - The response times of the jobs in T_2: **0.8**, 0.3, 0.2, 0.2, 0.8, ….
  - The response times of the jobs in T_3: **2**, 1.8, 2, 2, ….



Example of Critical Instants (2/2)

- $T_1 = (2, 0.6)$, $T_2 = (1, 2.5, 0.2)$, $T_3 = (3, 1.2)$
  - The response times of the jobs in T_2: 0.2, 0.2, **0.8**, 0.3, 0.2, ….
  - The response times of the jobs in T_3: 2, 2, **2**, 1.8, ….



Another Reason for Critical Instants

- Whenever release-time jitters are not negligible, the information on release times cannot be used to determine whether any algorithm can feasibly schedule the given system of tasks.
- Under this circumstance, we have no choice but to judge a fixed-priority algorithm according to its performance for tasks that are in phase because all fixed-priority algorithms have their worst-case performance for this combination of phases.

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

Time-Demand Analysis

- To determine whether a task can meet all its deadlines:
  - Compute the total demand for processor time by a job released at a critical instant of the task and by all the higher-priority tasks as a function of time from the critical instant.
  - Check whether this demand can be met before the deadline of the job.
- We consider one task at a time, starting from the task T_1 with the highest priority in order of decreasing priority.

Step by Step (1/2)

Assume all the tasks with higher priorities than T_i are schedulable, we want to determine whether the task T_i is schedulable.

1. Suppose the release time t_0 of the job is a critical instant of T_i.
2. At time $t_0 + t$ for $t \geq 0$, the total (processor) time demand $w_i(t)$ of this job and all the higher-priority jobs released in $[t_0, t]$ is given by

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \lceil t / p_k \rceil \cdot e_k \quad \text{for } 0 < t \leq p_i$$

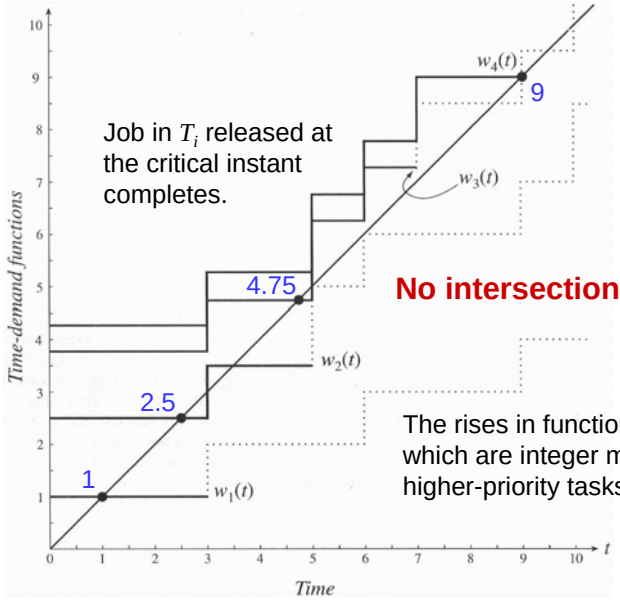
Supply of processor time: t
Demand of processor time in the interval: $w_i(t)$

Step by Step (2/2)

3. If $w_i(t) \leq t$ for some $t \leq D_i$, where D_i is equal to or less than p_i, all jobs in T_i can complete by their deadline.
4. If $w_i(t) > t$ for all $0 < t \leq D_i$, this job cannot complete by its deadline; T_i, and hence the given system of tasks, cannot be feasibly scheduled by the given fixed-priority algorithm.

- *Note that if the given tasks have known phases and periods and the jitters in release times are negligibly small, T_i may nevertheless be schedulable even though the time-demand analysis test indicates that it is not.*

Illustration of Time Demand Analysis



$T_1 = (\phi_1, 3, 1)$
$T_2 = (\phi_2, 5, 1.5)$
$T_3 = (\phi_3, 7, 1.25)$
$T_4 = (\phi_4, 9, 0.5)$

No intersection if unschedulable!

The rises in function occur at time instants which are integer multiples of periods of higher-priority tasks.

Time Demand Analysis Method (1/2)

- If we are not interested in the values of its maximum possible response time, only whether a task is schedulable, it suffices for us to check whether the time-demand function of the task is equal to or less than the supply at these instants.

Time Demand Analysis Method (2/2)

- Check whether the inequality

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \lceil t / p_k \rceil \cdot e_k \le t$$

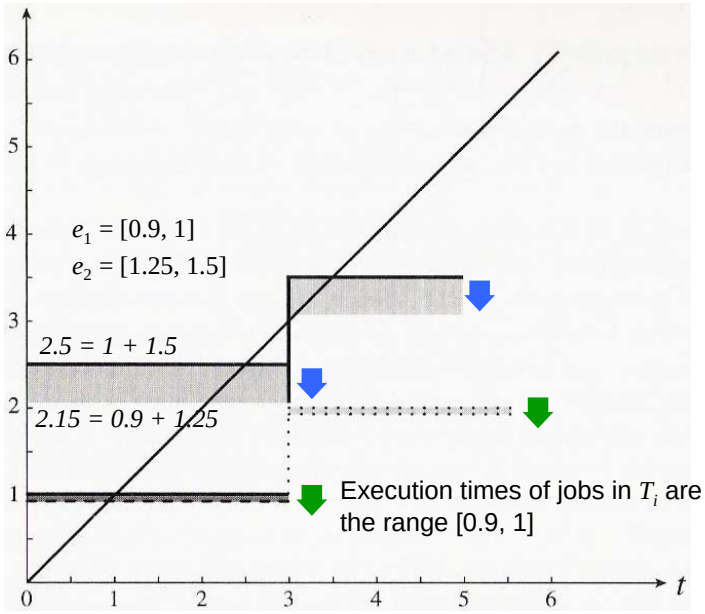
  is satisfied for values of t that are equal to

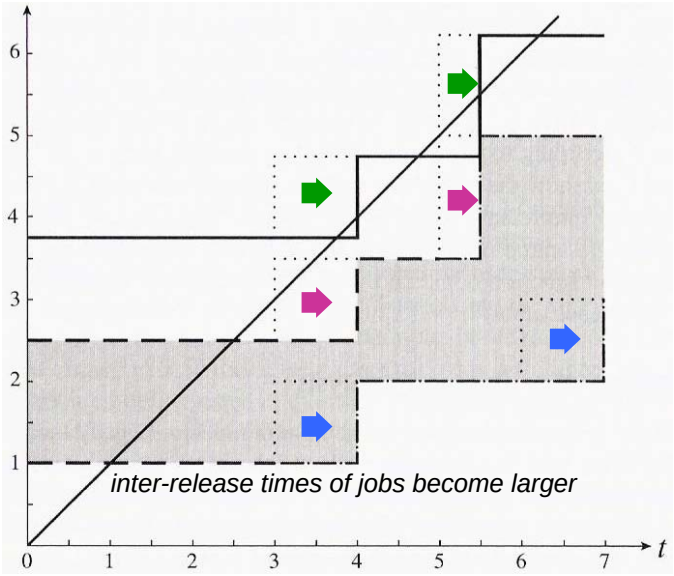
$$t = jp_k;\ k = 1, 2, ..., i;\ j = 1, 2, ..., \lfloor \min(p_i, D_i) / p_k \rfloor$$

  If this inequality is satisfied at any of these instants, T_i is schedulable.
- The time complexity of the time-demand analysis for each task is $O(np_n/p_1)$.

Robustness of the Time-Demand Analysis Method

- The conclusion that a task T_i is schedulable remains correct when
  - *the execution times of jobs may be less than their maximum execution times* and
  - *inter-release times of jobs may be larger than their respective periods*.





Outline

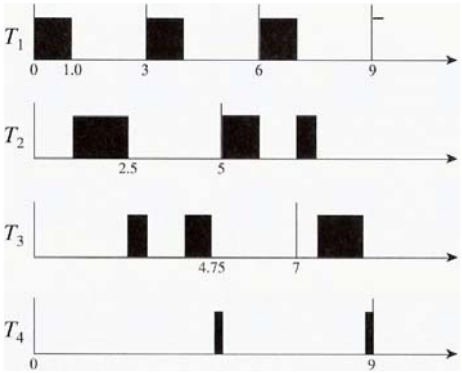
- A Schedulability Test for Fixed-Priority Tasks with Short Response Times:
  1. Critical Instants
  2. Time-Demand Analysis
  3. ***Alternatives to Time-Demand Analysis***

Alternative of Time-Demand Analysis

- We can determine whether a system of independent preemptable tasks is schedulable by simply *simulating the condition and observing whether the system is then schedulable*.
  - A way to test the schedulability of such a system is to construct a schedule of it according to the given scheduling algorithm
  - As long as Theorem 5 holds, it suffices for us to construct only the initial segment of length equal to the largest period of the tasks.

Worst-Case Simulation Method

- Assumptions:
  - The tasks are in phase
  - The actual execution times and inter-release times of jobs in each task T_i are equal to e_i and p_i, respectively.



A worst-case schedule of
- $T_1 = (3, 1)$
- $T_2 = (5, 1.5)$
- $T_3 = (7, 1.25)$
- $T_4 = (9, 0.5)$

The time complexity: $O(np_n/p_1)$.

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
- Practical Factors

Introduction

- This section describes a general time-demand analysis method to determine the schedulability of tasks whose *relative deadlines are larger than their respective periods*.
  - Since the response time of a task may be larger than its period, it may have *more than one job ready for execution at any time*.
- **References**
  - Lehoczky, J. P., "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines," Proceedings of the IEEE Real-Time Systems Symposium, December 1990.

Busy Intervals (1/2)

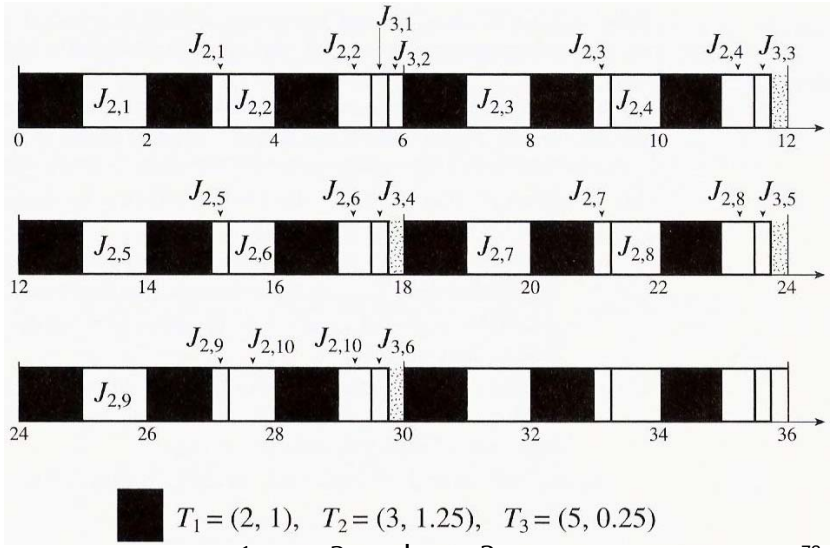
- A *level-π_i busy interval* $(t_0, t]$ begins at an instant t_0 when
  1. All jobs in $\mathbf{T}_i$ released before the instant have completed.
  2. A job in $\mathbf{T}_i$ is released.

  The interval ends at the first instant t after t_0 when all the jobs in $\mathbf{T}_i$ released since t_0 are complete.

Busy Intervals (2/2)

- In the interval $(t_0, t]$, the processor is busy all the time executing jobs with priorities π_i or higher, all the jobs executed in the busy interval are released in the interval, and at the end of the interval there is no backlog of jobs to be executed afterwards.
- **Definition.** We say that a level-π_i busy interval is in phase if the first jobs of all tasks that have priorities equal to or higher than priority π_i and are executed in this interval have the same release time.

Example Illustrating Busy Intervals (1/2)



$\pi_1 = 1, \pi_2 = 2,$ and $\pi_3 = 3$

Example Illustrating Busy Intervals (2/2)

- Every level-1 busy interval always ends 1 unit time after it begins.
- For this system, all the level-2 busy intervals are in phase.
  - They begin at times 0, 6, and so on which are the least common multiples of the periods of tasks T_1 and T_2.
  - Why? How about 3, 9, and so on?
  - The length of these intervals are all equal to 5.5.
- The second level-3 busy interval begins at time 6. (not in phase!)

General Schedulability Test

- It still suffices to confine our attention to the special case where the tasks are in phase.
- However, the first job $J_{i,1}$ may no longer have the largest response time among all jobs in T_i.
  - Consider T_1=(70,26) and T_2=(100,62): seven jobs of T_2 execute in the first level-2 busy interval with response times = 114, 102, 116, 104, 118, 106, and 94.
- We must examine **all the jobs** of T_i that are executed in the first level-π_i busy interval.
- If the response times of all these jobs are no greater than the relative deadline of T_i, T_i is schedulable; otherwise, T_i may not be schedulable.

General Time-Demand Analysis Method (1/3)

- Test one task at a time starting from the highest priority task T_1 in order of decreasing priority.
- For the purpose of determining whether a task T_i is schedulable, assume that all the tasks are in phase and the first level-π_i busy interval begins at time 0.
- While testing whether all the jobs in T_i can meet their deadlines (i.e., whether T_i is schedulable), consider the subset $\mathbf{T}_i$ of tasks with priorities π_i or higher.

General Time-Demand Analysis Method (2/3)

1) If the first job of every task in $\mathbf{T}_i$ *completes by the end of the first period of the task*, check whether the first job $J_{i,1}$ in T_i meets its deadline. T_i is schedulable if $J_{i,1}$ completes in time. Otherwise, T_i is not schedulable.
2) If the first job of some task in $\mathbf{T}_i$ ***DOES NOT*** *complete by the end of the first period of the task*, do the following:

General Time-Demand Analysis Method (3/3)

a) Compute the length of the in phase level-π_i busy interval by solving the equation $t = \sum_{k=1}^{i} \lceil t / p_k \rceil \cdot e_k$ iteratively, starting from $t^{(1)} = \sum_{k=1}^{i} e_k$ until $t^{(l+1)} = t^{(l)}$ for some $l \geq 1$. The solution $t^{(l)}$ is the length of the level-π_i busy interval.
b) Compute the maximum response times of all $\lceil t^{(l)} / p_i \rceil$ jobs of T_i in the in-phase level-π_i busy interval in the manner described below and determine whether they complete in time.

T_i is schedulable if all these jobs complete in time; otherwise T_i is not schedulable.

Time-Demand Function $w_{i,1}(t)$ (1/2)

- The response time of the first job $J_{i,1}$ of T_i in the first in-phase level-π_i busy interval is similar to the time-demand function in Page 58.
- An important difference is that the expression remains valid for all $t > 0$ before the end of the busy interval.

$$w_{i,1}(t) = e_i + \sum_{k=1}^{i-1} \lceil t / p_k \rceil \cdot e_k \qquad \text{for } 0 < t \leq w_{i,1}(t)$$

- The maximum possible response time $W_{i,1}$ of $J_{i,1}$ is equal to the smallest value of t that satisfies the equation $t = w_{i,1}(t)$.

Time-Demand Function $w_{i,1}(t)$ (2/2)

- To obtain the maximum possible response time $W_{i,1}$ of $J_{i,1}$, we solve the equation iteratively and terminate the iteration only when we find $t^{(l+1)}$ equal to $t^{(l)}$.
- Because U_i is no greater than 1, this equation always has a finite solution, and the solution can be found after a finite number of iterations.

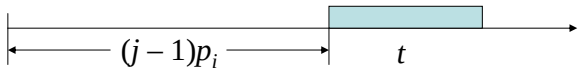
Time-Demand Function $w_{i,j}(t)$

- **Lemma 6.** The maximum response time $W_{i,j}$ of the j-th job of T_i in an in-phase level-π_i busy interval is equal to the smallest value of t that satisfies the equation

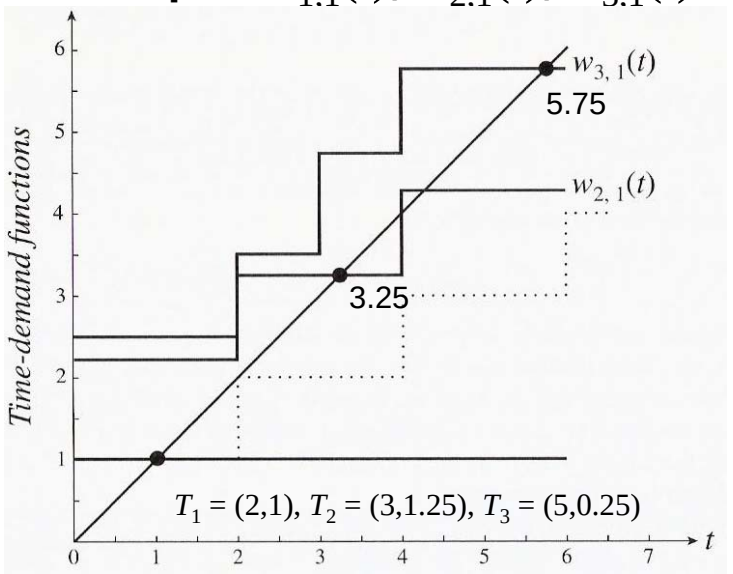
$$t = w_{i,j}(t + (j-1)p_i) - (j-1)p_i$$

where $w_{i,j}(\ )$ is given by

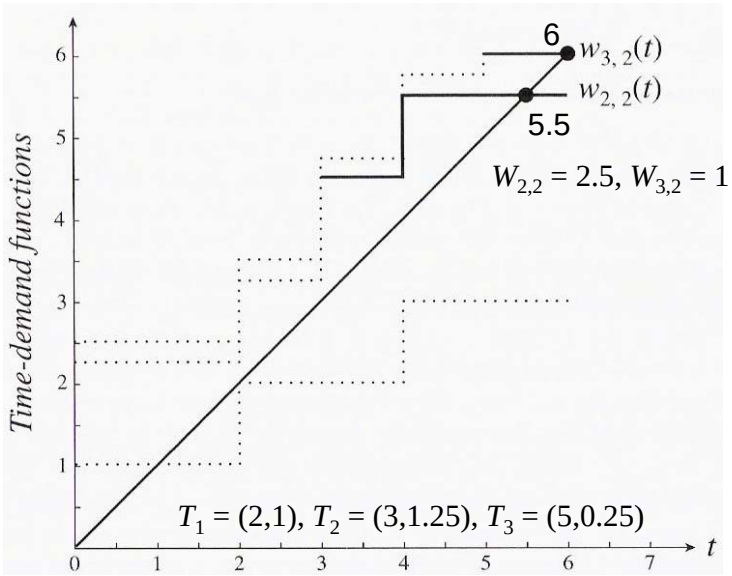
$$w_{i,j}(t) = je_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \quad \text{for } (j-1)p_i < t \le w_{i,j}(t)$$



Example: $w_{1,1}(t)$, $w_{2,1}(t)$, $w_{3,1}(t)$



Example: $w_{2,2}(t)$, $w_{3,2}(t)$



Example: Find $W_{2,2}$

$T_1 = (2,1)$, $T_2 = (3,1.25)$, $T_3 = (5,0.25)$

- $t = w_{i,j}(t + (j-1)p_i) - (j-1)p_i$
- $w_{i,j}(t) = je_i + \sum_{k=1}^{i-1} \lceil t / p_k \rceil \cdot e_k$ for $(j-1)p_i < t \le w_{i,j}(t)$

$$t = 2 \times 1.25 + \lceil (t+3)/2 \rceil - 3$$

Substitute $t^{(1)} = 1.25$ on the right hand side of the equation.

We obtain: $W_{2,2} = 2.5$

How about $W_{3,2}$?

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*
- Practical Factors

Motivation

- Time-demand analysis method requires the *periods* and *execution times* of all the tasks in an application system to determine whether the system is schedulable.
- Before we have completed the design of the application system, some of these parameters may not be known.
- It is desirable to have a schedulability condition similar to Theorem 1 and 2 for the EDF and LST algorithms.

Outline

- Sufficient Schedulability Conditions for the RM and DM Algorithms
  1. ***Schedulable Utilization of the RM Algorithm for Tasks with $D_i = p_i$.***
  2. Schedulable Utilization of RM Algorithms as Functions of Task Parameters
  3. Schedulable Utilization of Fixed Priority Tasks with Arbitrary Relative Deadlines
  4. Schedulable Utilization of the RM Algorithm for Multiframe Tasks

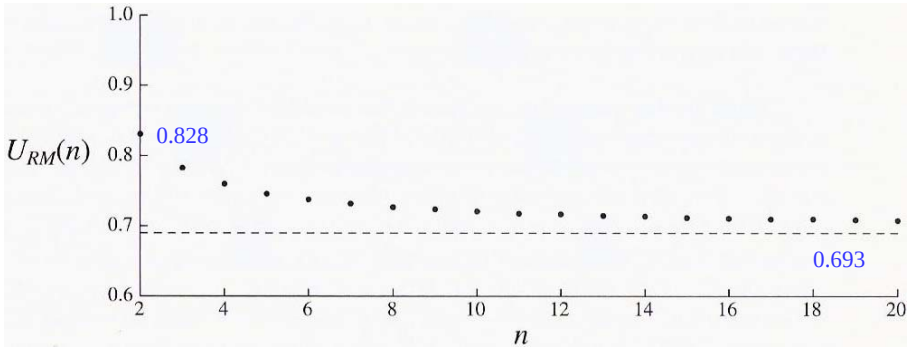
Schedulable Utilization of the RM ($D_i = p_i$)

- **Theorem 7.** A system of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U is less than or equal to

$$U_{RM}(n) = n(2^{1/n} - 1)$$

- $U_{RM}(n)$ is the schedulable utilization of the RM algorithm when $D_i = p_i$ for all $1 \le i \le n$.

$U_{RM}(n)$ as a Function of n



- As long as the total utilization of a system satisfies $U(n) \leq U_{RM}(n)$, it will never miss any deadline.
- We can reach this conclusion without considering the individual values of the phases, periods, and execution times.

Example 1

- Does the system $\mathbf{T} = \{(1.0, 0.25), (1.25, 0.1), (1.5, 0.3), (1.75, 0.07), (2.0, 0.1)\}$ schedulable?

Example 2

- Does the system $\mathbf{T} = \{(0.3, 1.3, 0.1), (1.0, 1.5, 0.3), (1.75, 0.1), (2.0,0.1), (7.0, 2.45)\}$ schedulable?

Schedulable Utilization of the RM

- Since $U(n) \leq U_{RM}(n)$ is not a necessary condition, a system of tasks may nevertheless be schedulable even when its total utilization exceeds the schedulable bound!
- **Example:** Total utilization of $\mathbf{T} = \{(3, 1), (5, 1.5), (7, 1.25), (9, 0.5)\}$ is 0.85, which is larger than $U_{RM}(4) = 0.756$, but this system is schedulable according to the RM algorithm!

Outline

- Sufficient Schedulability Conditions for the RM and DM Algorithms
  1. Schedulable Utilization of the RM Algorithm for Tasks with $D_i = p_i$.
  2. ***Schedulable Utilization of RM Algorithms as Functions of Task Parameters***
  3. Schedulable Utilization of Fixed Priority Tasks with Arbitrary Relative Deadlines
  4. Schedulable Utilization of the RM Algorithm for Multiframe Tasks

Enhanced Schedulable Utilization

- When some of the task parameters are known, this information allows us to improve the schedulable utilization of the RM algorithm:
  - The Utilization of Individual Tasks
  - The Number n_h of Disjoint Subsets Each Containing Simply Periodic Tasks
  - Some Functions of the Periods of the Tasks

Task Utilizations

- **Corollary 8.** n independent, preemptable periodic tasks with relative deadlines equal to their respective periods are schedulable rate-monotonically if their utilizations $u_1, u_2, \ldots, u_n$ satisfy the inequality

$$(1+u_1)(1+u_2)...(1+u_n) \leq 2$$

Subsets of Simply Periodic Tasks (1/2)

- **Example.** We can partition the system $\mathbf{T}$ of tasks with periods 4, 7, 8, 14, 16, 28, 32, 56, and 64 into two subsets $\mathbf{Z}_1$ and $\mathbf{Z}_2$. $\mathbf{Z}_1$ contains the tasks with periods 4, 8, 16, 32, and 64; and $\mathbf{Z}_2$ contains tasks with periods 7, 14, 28, and 56. Let $U(\mathbf{Z}_1)$ and $U(\mathbf{Z}_2)$ denote the total utilization of the tasks in $\mathbf{Z}_1$ and $\mathbf{Z}_2$, respectively.
- ➡ If $U(\mathbf{Z}_1) + U(\mathbf{Z}_2) \leq 0.828$, all these tasks are schedulable rate-monotonically.

References

Kuo, T. W. and A. K. Mok, "Load Adjustment in Adaptive Real-Time Systems," Proceedings of the IEEE Real-Time Systems Symposium, December 1991.

Subsets of Simply Periodic Tasks (2/2)

- **Theorem 9.** If a system $\mathbf{T}$ of independent, preemptable periodic tasks, whose relative deadlines are equal to their respective periods, can be partitioned into n_h disjoint subsets, $\mathbf{Z}_1, \mathbf{Z}_2, ..., \mathbf{Z}_{n_h}$, each of which contains simply periodic tasks, then the system is schedulable rate-monotonically if the total utilizations $U(\mathbf{Z}_i)$, for $i = 1, 2, \ldots, n_h$, of the tasks in the subsets satisfy the inequality

$$(1+U(\mathbf{Z}_1))(1+U(\mathbf{Z}_2))...(1+U(\mathbf{Z}_{n_h})) \leq 2$$

Task Periods (1/2)

- For a given system $\mathbf{T}$ of tasks, $\mathbf{T'}$ is an accelerated set of $\mathbf{T}$ if following properties hold:
  1. There is a task T_i' in $\mathbf{T'}$ if and only if there is a task T_i in $\mathbf{T}$.
  2. The execution time of T_i' is equal to the execution time of T_i.
  3. The period of T_i' is shorter than the period of T_i.

Task Periods (2/2)

- **Theorem 10.** A system $\mathbf{T}$ of independent, preemptive periodic tasks whose relative deadlines are equal to their respective periods is schedulable according the RM algorithm if it has an accelerated set $\mathbf{T'}$ which is simply periodic and has a total utilization equal to or less than 1.

Outline

- Sufficient Schedulability Conditions for the RM and DM Algorithms
  1. Schedulable Utilization of the RM Algorithm for Tasks with $D_i = p_i$.
  2. Schedulable Utilization of RM Algorithms as Functions of Task Parameters
  3. ***Schedulable Utilization of Fixed Priority Tasks with Arbitrary Relative Deadlines***
  4. Schedulable Utilization of the RM Algorithm for Multiframe Tasks

Observation

1. A system of n tasks with a total utilization $U_{RM}(n)$ may not be schedulable rate-monotonically when the relative deadlines of some tasks are shorter than their periods.
2. If the relative deadlines of the tasks are larger than the respective task periods, we expect the schedulable utilization of the RM algorithm to be larger than $U_{RM}(n)$.

➡ We now consider the case where the relative deadline D_i of every task is equal to δ times its period p_i for some $0 < \delta$.

Arbitrary Relative Deadlines

- **Theorem 11.** A system of n independent, preemptable periodic tasks with relative deadlines $D_i = \delta p_i$ for all $1 \leqq i \leqq n$ is schedulable rate-monotonically if its total utilization is equal to or less than

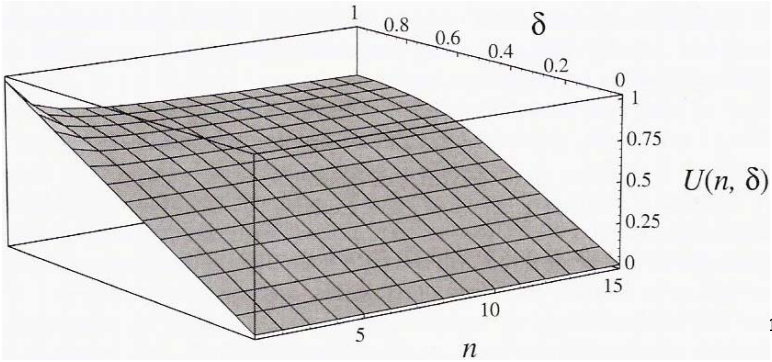
$$U_{RM}(n, \delta) = \delta(n-1)\left[\left(\frac{\delta+1}{\delta}\right)^{1/n-1} - 1\right], \quad \text{for } \delta = 2, 3, \ldots$$

$$= n((2\delta)^{1/n} - 1) + 1 - \delta, \quad \text{for } 0.5 \leqq \delta \leqq 1$$

$$= \delta, \quad \text{for } 0 \leqq \delta \leqq 0.5$$

$U_{RM}(n, \delta)$

n	δ =4.0	δ =3.0	δ =2.0	δ =1.0	δ =0.9	δ =0.8	δ =0.7	δ =0.6	δ =0.5
2	0.944	0.928	0.898	0.828	0.783	0.729	0.666	0.590	0.500
3	0.926	0.906	0.868	0.779	0.749	0.708	0.656	0.588	0.500
4	0.917	0.894	0.853	0.756	0.733	0.698	0.651	0.586	0.500
5	0.912	0.888	0.844	0.743	0.723	0.692	0.648	0.585	0.500
∞	0.892	0.863	0.810	0.693	0.687	0.670	0.636	0.582	0.500



Outline

- Sufficient Schedulability Conditions for the RM and DM Algorithms
  1. Schedulable Utilization of the RM Algorithm for Tasks with $D_i = p_i$.
  2. Schedulable Utilization of RM Algorithms as Functions of Task Parameters
  3. Schedulable Utilization of Fixed Priority Tasks with Arbitrary Relative Deadlines
  4. ***Schedulable Utilization of the RM Algorithm for Multiframe Tasks***

Motivation

- Consider a task that models the transmission of an MPEG compressed video over a network link.
  - Jobs in this task, modeling the transmissions of individual video frames, are released periodically.
  - The size of I-frames can be very large compared with that of B- and P- frames, the execution times of the jobs can vary widely.
  - When modeled as a periodic task, the execution time of the task is equal to the transmission time of an I-frame.
- If we were to determine whether a system of such tasks is schedulable based on the schedulability tests we have learned, we would surely underutilized the processor.
- The multiframe task model is a more accurate model and leads to more accurate schedulability tests.

Multiframe Task Model

- Each task T_i is characterized by a 4-tuple $(p_i, \xi_i, e_i^p, e_i^n)$
  - p_i : period of the task

  Jobs in T_i have either one of two possible maximum execution time
  - e_i^p: peak execution time
  - e_i^n: normal execution time

  Each period which begins at the release time of a job with the peak execution time is called a **peak frame**, and the other periods are called **normal frames**.
  - ξ_i : each peak frame is followed by $\xi_i - 1$ normal frames, which in turn are followed by a peak frame an so on.

References Mok, A. K.-L., and D. Chen, "A Multiframe Model for Real-Time Tasks," Proceedings of IEEE Real-Time Systems Symposium, December 1996.

Example

- The task (33, 6, 1.0, 0.3) can model an MPEG video transmission task.
  - The period of the task is 33ms.
  - The execution time of the job in each peak frame, which models the transmission of an I-frame in the video stream, is never more than 1ms.
  - I-frame is followed by 5 normal frames.
  - The execution times of jobs released in normal frames are never more than 0.3. These jobs model the transmissions of B- and P-frames in the video.
  - They are followed by the transmission of an I-frame, that is, a peak frame, which is in turn followed by 5 normal frames, and so on.

Critical Instant and Load Variation

- Critical Instant

  The response time of a job $J_{i,k}$ in T_i has the maximum possible value if the k-th period, which begins when $J_{i,k}$ is released, is a peak frame and this peak frame begins at the same time as a peak frame in every high-priority task.
- Load Variation

  Given a system of n multiframe tasks, the load variation, denoted by Ξ, of the system is $\min_{1 \le i \le n}(e_i^p / e_i^n)$.

Schedulable Utilization

- **Theorem 12.** A system of n independent, preemptable multiframe tasks, whose relative deadlines are equal to the respective periods, is schedulable according to the RM algorithm if their total utilization is no greater than

$$U_{RM}(n,\Xi) = \Xi\, n\left(\left(\frac{\Xi+1}{\Xi}\right)^{1/n} - 1\right)$$

$U_{RM}(n, \Xi)$