Sharif University of Technology
Department of Computer Science and Engineering

Lec. 4:
Periodic Task Scheduling

Real-Time Computing

S. Safari

2023

Introduction

- In many real-time control applications, periodic activities represent the major computational demand in the system.
- Periodic tasks typically arise from sensory data acquisition, control loops, action planning, and system monitoring.
- Such activities need to be cyclically executed at specific rates, which can be derived from the application requirements.
- Basic algorithms:
  - Timeline Scheduling
  - Rate Monotonic
  - Earliest Deadline First
  - Deadline Monotonic

Notations

Γ	denotes a set of periodic tasks;
τ_i	denotes a generic periodic task;
$\tau_{i,j}$	denotes the jth instance of task τ_i;
$r_{i,j}$	denotes the release time of the jth instance of task τ_i;
Φ_i	denotes the *phase* of task τ_i; that is, the release time of its first instance ($\Phi_i = r_{i,1}$);
D_i	denotes the relative deadline of task τ_i;
$d_{i,j}$	denotes the absolute deadline of the jth instance of task τ_i ($d_{i,j} = \Phi_i + (j-1)T_i + D_i$).
$s_{i,j}$	denotes the start time of the jth instance of task τ_i; that is, the time at which it starts executing.
$f_{i,j}$	denotes the finishing time of the jth instance of task τ_i; that is, the time at which it completes the execution.

Hypotheses for Schedulability Analysis

A1. The instances of a periodic task τ_i are regularly activated at a constant rate. The interval T_i between two consecutive activations is the *period* of the task.

A2. All instances of a periodic task τ_i have the same worst-case execution time C_i.

A3. All instances of a periodic task τ_i have the same relative deadline D_i, which is equal to the period T_i.

A4. All tasks in Γ are independent; that is, there are no precedence relations and no resource constraints.

A5. No task can suspend itself, for example on I/O operations.

A6. All tasks are released as soon as they arrive.

A7. All overheads in the kernel are assumed to be zero.

Some Parameters for Periodic Tasks

- **Hyperperiod**
  - The minimum interval of time after which the schedule repeats itself.
  - For a set of periodic tasks synchronously activated at time t=0, the hyperperiod is given by the least common multiple of the periods:

$$H = lcm(T_1, \ldots, T_n).$$

- **Job response time**
  - The time (measured from the release time) at which the job is terminated:

$$R_{i,k} = f_{i,k} - r_{i,k}.$$

Some Parameters for Periodic Tasks (2)

- **Task response time**
  - The maximum response time among all the jobs:

$$R_i = \max_k R_{i,k}.$$

- **Critical instant** of a task
  - The arrival time that produces (have) the largest task response time.
- **Critical time zone** of a task
  - The interval between the critical instant and the response time of the corresponding request of the task

Some Parameters for Periodic Tasks (3)

- **Relative Start Time Jitter** of a task
  - The maximum deviation of the start time of two consecutive instances:

$$RRJ_i = \max_k |(s_{i,k} - r_{i,k}) - (s_{i,k-1} - r_{i,k-1})|.$$

- **Absolute Start Time Jitter** of a task
  - The maximum deviation of the start time among all instances:

$$ARJ_i = \max_k (s_{i,k} - r_{i,k}) - \min_k (s_{i,k} - r_{i,k}).$$

Some Parameters for Periodic Tasks (4)

- **Relative Finishing Jitter** of a task
  - The maximum deviation of the finishing time of two consecutive instances:

$$RFJ_i = \max_k |(f_{i,k} - r_{i,k}) - (f_{i,k-1} - r_{i,k-1})|.$$

- **Absolute Finishing Jitter** of a task
  - The maximum deviation of the finishing time among all instances:

$$AFJ_i = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k}).$$

Processor Utilization Factor

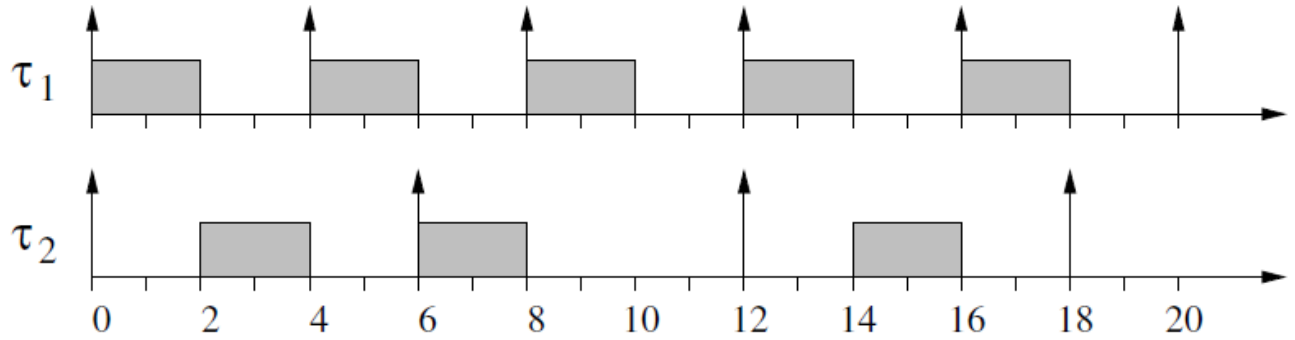
- Given a set Γ of n periodic tasks, the processor utilization factor U is the fraction of processor time spent in the execution of the task set.
- Since C_i/T_i is the fraction of processor time spent in executing task τ_i, the utilization factor for n tasks is given by:

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

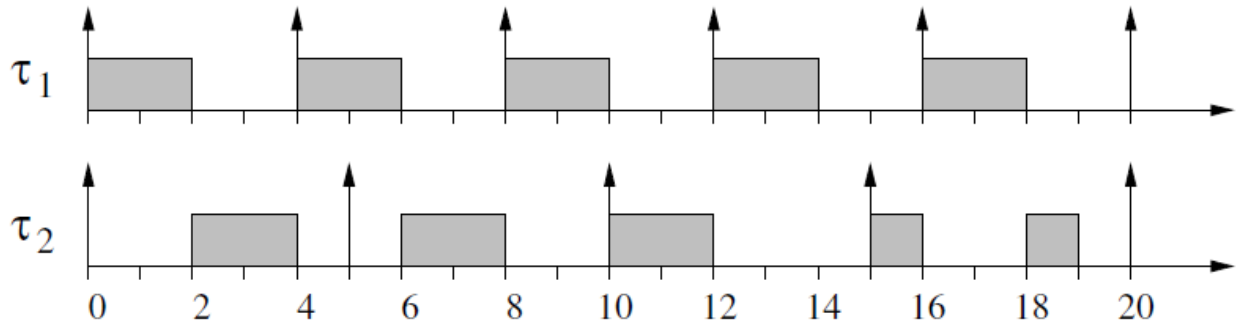
- The processor utilization factor provides a measure of the computational load on the CPU due to the periodic task set.
- Although the CPU utilization can be improved by increasing tasks' computation times or by decreasing their periods, there exists a maximum value of U below which Γ is schedulable and above which Γ is not schedulable.

An Example of Processor Utilization

- An example of two tasks (where τ_1 has higher priority than τ_2) in which $U_{upper\text{-}bound} = 2/4+2/6=5/6 = 0.833$



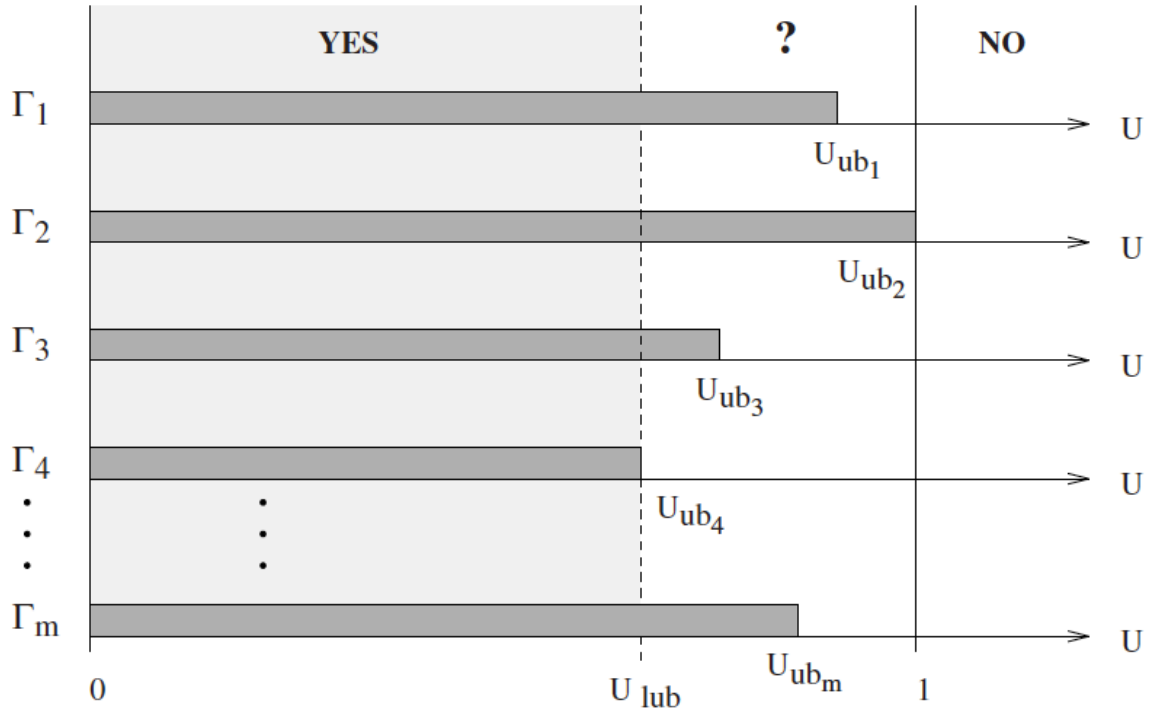
- Another example in which $U_{ub} = 0.9$. Notice that setting $T_1 = 4$ and $T_2 = 4$, $U_{upper\text{-}bound}$ becomes 1.0.



Least Upper Bound

- For a given algorithm *A*, the least upper bound $U_{lub}(A)$ of the processor utilization factor is the minimum of the utilization factors over all task sets that fully utilize the processor:

$$U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A).$$



Least Upper Bound (2)

- If the utilization factor of a task set is greater than 1.0, the task set cannot be scheduled by any algorithm.
- To show this result, let H be the hyperperiod of the task set. If $U>1$, we also have $UH > H$, which can be written as:

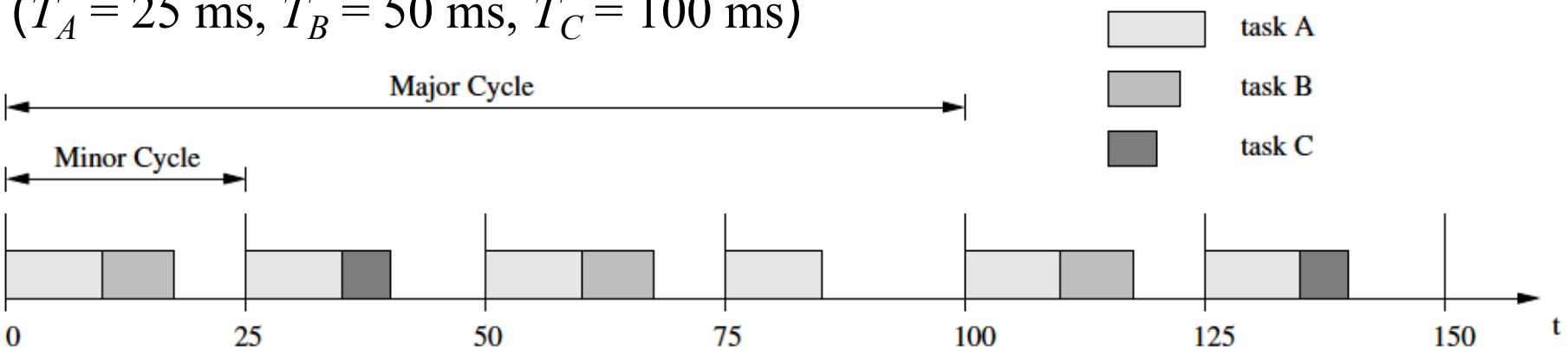
$$\sum_{i=1}^{n} \frac{H}{T_i} C_i \; > \; H.$$

- The factor (H/T_i) represents the (integer) number of times τ_i (the number of jobs) is executed in the hyperperiod, whereas the quantity $(H/T_i)C_i$ is the total computation time requested by τ_i in the hyperperiod.
- The sum on the left hand side represents the total computational demand requested by the task set in $[0,H)$.
- Clearly, if the total demand exceeds the available processor time, there is no feasible schedule for the task set.

Timeline scheduling

- **Timeline Scheduling (TS)**, also known as a *Cyclic Executive*, is one of the most used approaches to handle periodic tasks in defense military systems and traffic control systems.
- The method consists of dividing the temporal axis into slots of equal length.
  - One or more tasks can be allocated for execution, in such a way to respect the frequencies derived from the application requirements.

$(T_A = 25 \text{ ms}, T_B = 50 \text{ ms}, T_C = 100 \text{ ms})$



Advantages of Timeline scheduling

- The main advantage of timeline scheduling is its simplicity.
- Can be implemented by programming a timer to interrupt with a period equal to the minor cycle and by writing a main program that calls the tasks in the order given in the major cycle, inserting a time synchronization point at the beginning of each minor cycle.
- Since the task sequence is not decided by a scheduling algorithm in the kernel, but it is triggered by the calls made by the main program, there are no context switches, so the runtime overhead is very low.

Disadvantages of Timeline scheduling

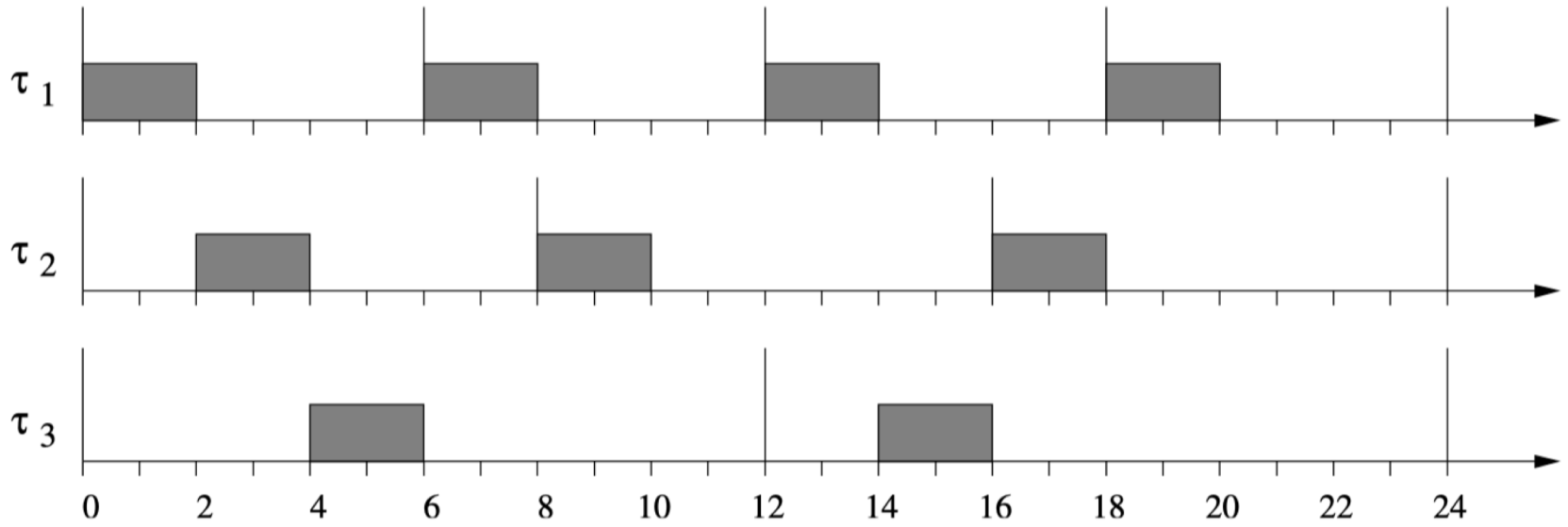
- Very fragile during overload conditions
- If a task does not terminate at the minor cycle boundary, it can either be continued or aborted.
  - The system may run into a critical situation. In fact, if the failing task is left in execution, it can cause a domino effect on the other tasks, breaking the entire schedule (timeline break).
  - If the failing task is aborted while updating some shared data, the system may be left in an inconsistent state, jeopardizing the correct system behavior.
- Another big problem of the timeline scheduling technique is its sensitivity to application changes. If updating a task requires an increase of its computation time or its activation frequency, the entire scheduling sequence may need to be reconstructed from scratch.

Rate Monotonic Scheduling

- The **Rate Monotonic (RM)** scheduling algorithm is a simple rule that assigns priorities to tasks according to their request rates.
- Tasks with higher request rates (shorter periods) will have higher priorities.
- Since periods are constant, RM is a fixed-priority assignment.
  - A priority P_i is assigned to the task before execution and does not change over time.
- RM is intrinsically preemptive.
  - The currently executing task is preempted by a newly arrived task with shorter period.
- In 1973, Liu and Layland [LL73] showed that RM is optimal among all fixed-priority assignments in the sense that no other fixed-priority algorithms can schedule a task set that cannot be scheduled by RM.

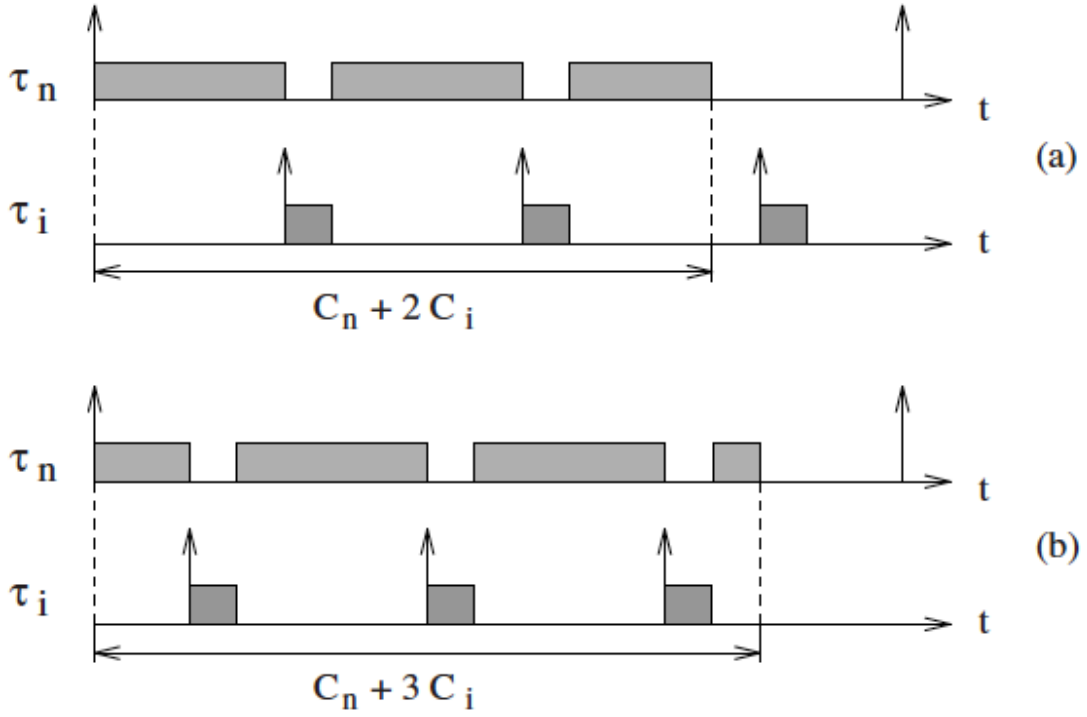
Example of Rate Monotonic Scheduling

	C_i	T_i
τ_1	2	6
τ_2	2	8
τ_3	2	12



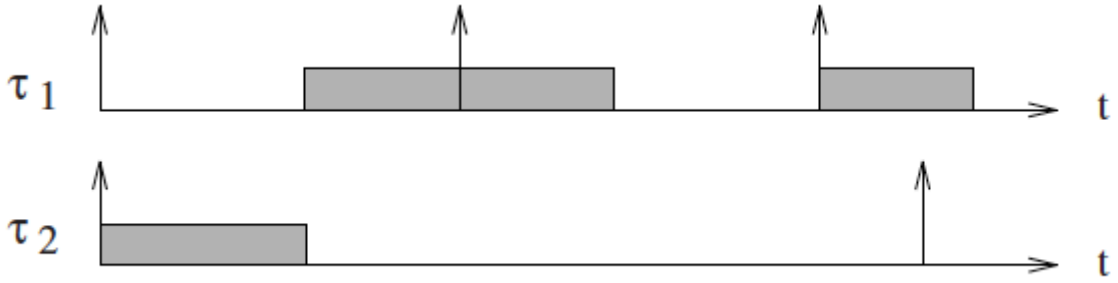
Optimality of Rate Monotonic Scheduling

- Let $\Gamma = \{\tau_1, \tau_2, \ldots, \tau_n\}$ be the set of periodic tasks ordered by increasing periods, with τ_n being the task with the longest period.
  - According to RM, τ_n will also be the task with the lowest priority.



Optimality of Rate Monotonic Scheduling (2)

- Consider a set of two periodic tasks τ_1 and τ_2, with $T_1 < T_2$.
- If priorities are not assigned according to RM, then task τ_2 will receive the highest priority.

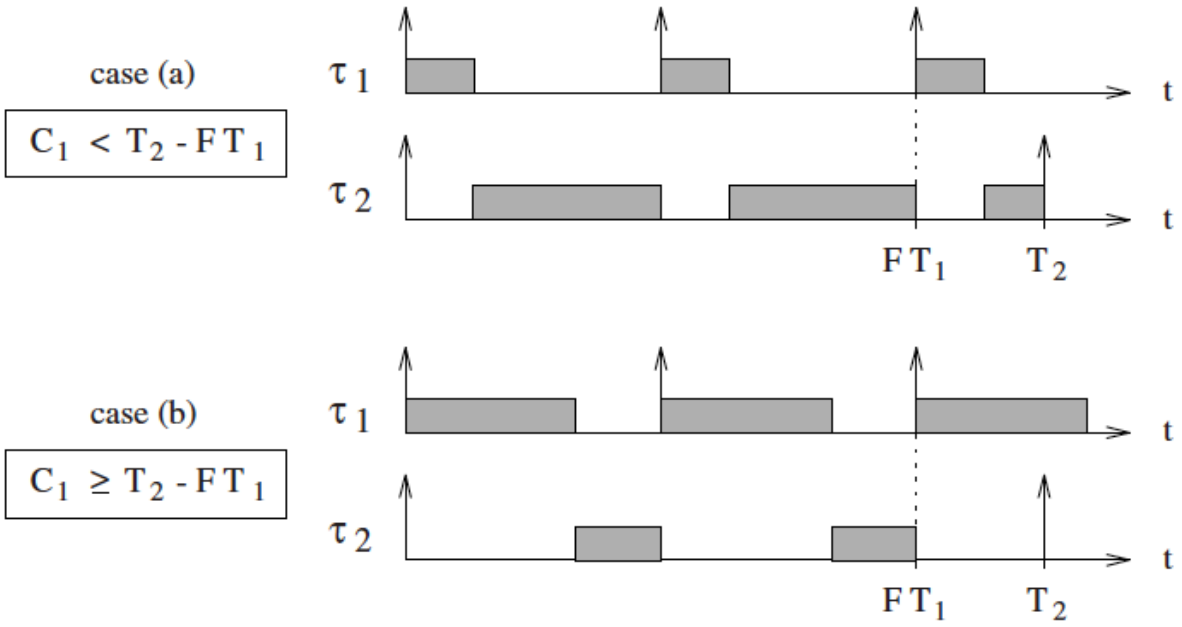


- The schedule is feasible if the following inequality is satisfied:

$$C_1 + C_2 \leq T_1.$$

Optimality of Rate Monotonic Scheduling (3)

- If priorities are assigned according to RM, task T_1 will receive the highest priority.
  - In order to guarantee a feasible schedule two cases must be considered.
- Let $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$ be the number of periods of τ_1 entirely contained in T_2.



Case 1 in Optimality of RM

- The task set is schedulable if:

$$(F+1)C_1 + C_2 \leq T_2.$$

- By multiplying both sides of $C_1 + C_2 \leq T_1$ by *F* we obtain:

$$FC_1 + FC_2 \leq FT_1,$$

- and, since $F \geq 1$, we can write:

$$FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1.$$

- Adding C_1 to each member we get:

$$(F+1)C_1 + C_2 \leq FT_1 + C_1.$$

- Since we assumed that $C_1 < T_2 - FT_1$, we have:

$$(F+1)C_1 + C_2 \leq FT_1 + C_1 < T_2,$$

Case 2 in Optimality of RM

- The task set is schedulable if:

$$FC_1 + C_2 \leq FT_1.$$

- By multiplying both sides of $C_1 + C_2 \leq T_1$ by *F* we obtain:

$$FC_1 + FC_2 \leq FT_1,$$

- and, since $F \geq 1$, we can write:

$$FC_1 + C_2 \leq FC_1 + FC_2 \leq FT_1.$$

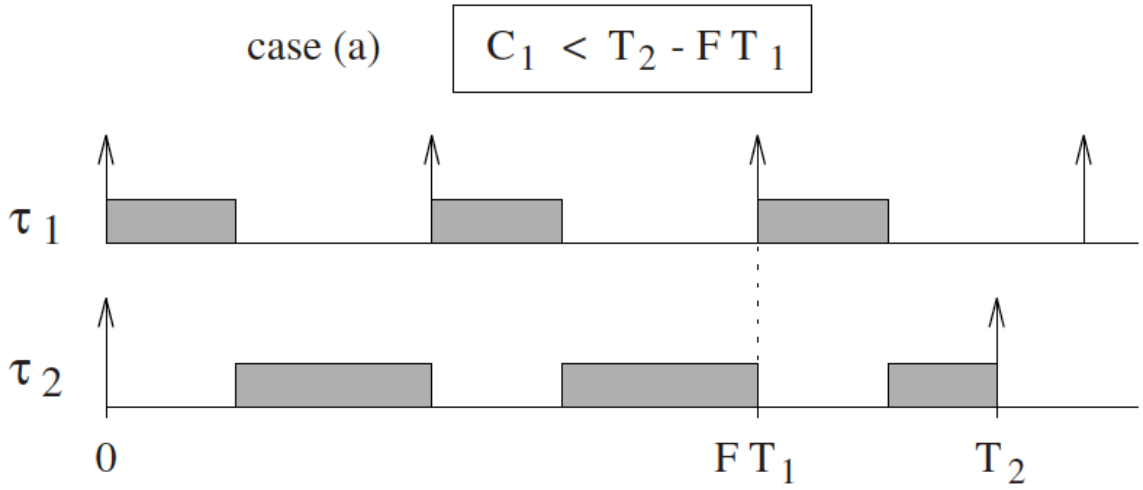
Basically, it has been shown that, given two periodic tasks τ_1 and τ_2, with $T_1 < T_2$, if the schedule is feasible by an arbitrary priority assignment, then it is also feasible by RM. That is, RM is optimal. This result can easily be extended to a set of *n* periodic tasks.

Least Upper Bound (U_{LUB}) of the Processor Utilization Factor for the RM

- Consider a set of two periodic tasks τ_1 and τ_2, with $T_1 < T_2$. In order to compute U_{lub} for RM, we have to:
  - Assign priorities to tasks according to RM, so that τ_1 is the task with the highest priority;
  - Compute the upper bound U_{ub} for the task set by inflating computation times to fully utilize the processor;
  - Minimize the upper bound U_{ub} with respect to all the other task parameters.
- Let $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$ be the number of periods of τ_1 entirely contained in T_2.
- Two cases must be considered.

Least Upper Bound (U_{LUB}) of RM (2)

- **Case 1**: The computation time of τ_1 (synchronously activated with τ_2) is short enough that all its requests are completed before the second request of τ_2. That is, $C_1 \leq T_2 - FT_1$.



- In this situation, the largest possible value of C2 is:

$$C_2 = T_2 - C_1(F + 1),$$

Least Upper Bound (U_{LUB}) of RM (3)

- The corresponding upper bound U_{ub} is:

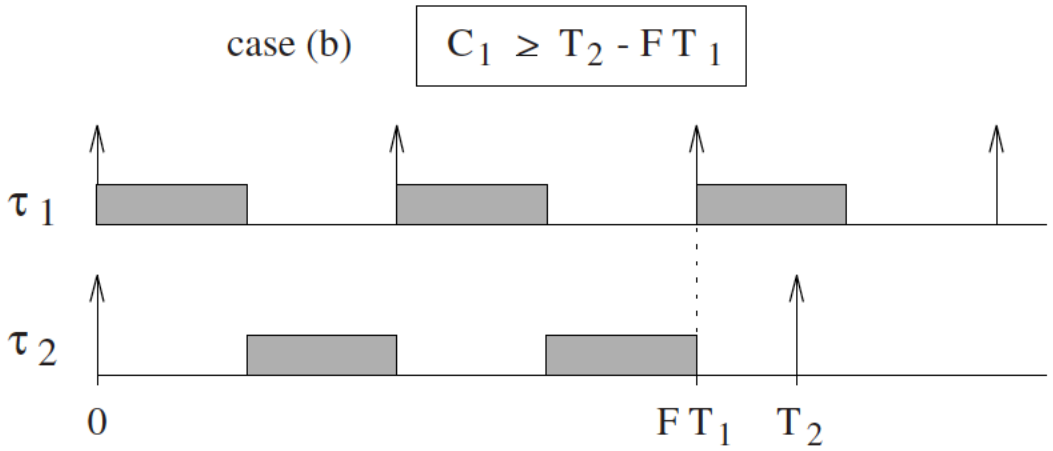
$$\begin{aligned} U_{ub} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{T_2 - C_1(F+1)}{T_2} = \\ &= 1 + \frac{C_1}{T_1} - \frac{C_1}{T_2}(F+1) = \\ &= 1 + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - (F+1)\right]. \end{aligned}$$

- Since the quantity in square brackets is negative, U_{ub} is monotonically decreasing in C_1, and, being $C_1 \leq T_2 - FT_1$, the minimum of U_{ub} occurs for:

$$C_1 = T_2 - FT_1.$$

Least Upper Bound (U_{LUB}) of RM (4)

- **Case 2**: The computation time of τ_1 (synchronously activated with τ_2) is long enough to overlap with the second request of τ_2. That is, $C_1 \geq T_2 - FT_1$.



- In this situation, the largest possible value of C_2 is:

$$C_2 = (T_1 - C_1)F,$$

Least Upper Bound (U_{LUB}) of RM (5)

- The corresponding upper bound U_{ub} is:

$$\begin{aligned} U_{ub} &= \frac{C_1}{T_1} + \frac{C_2}{T_2} = \frac{C_1}{T_1} + \frac{(T_1 - C_1)F}{T_2} = \\ &= \frac{T_1}{T_2}F + \frac{C_1}{T_1} - \frac{C_1}{T_2}F = \\ &= \frac{T_1}{T_2}F + \frac{C_1}{T_2}\left[\frac{T_2}{T_1} - F\right]. \end{aligned}$$

- Since the quantity in square brackets is positive, U_{ub} is monotonically increasing in C_1 and, being $C_1 \geq T_2 - F\mathrm{T}_1$, the minimum of U_{ub} occurs for:

$$C_1 = T_2 - FT_1.$$

- **In both cases, the minimum value of U_{ub} occurs for $C_1 = T_2 - T_1F$.**

Least Upper Bound (U_{LUB}) of RM (6)

- Using the minimum value of C_1, from U_{ub} in Case 2 we have:

$$\begin{aligned} U &= \frac{T_1}{T_2}F + \frac{C_1}{T_2}\left(\frac{T_2}{T_1} - F\right) = \\ &= \frac{T_1}{T_2}F + \frac{(T_2 - T_1F)}{T_2}\left(\frac{T_2}{T_1} - F\right) = \\ &= \frac{T_1}{T_2}\left[F + \left(\frac{T_2}{T_1} - F\right)\left(\frac{T_2}{T_1} - F\right)\right]. \end{aligned}$$

- To simplify the notation, let $G = T_2/T_1 - F$. Thus:

$$\begin{aligned} U &= \frac{T_1}{T_2}(F + G^2) = \frac{(F + G^2)}{T_2/T_1} = \\ &= \frac{(F + G^2)}{(T_2/T_1 - F) + F} = \frac{F + G^2}{F + G} = \\ &= \frac{(F + G) - (G - G^2)}{F + G} = 1 - \frac{G(1 - G)}{F + G}. \end{aligned}$$

Least Upper Bound (U_{LUB}) of RM (7)

- Since $0 \leq G < 1$, the term $G(1 - G)$ is nonnegative. Hence, U is monotonically increasing with F. As a consequence, the minimum of U occurs for the minimum value of F; namely, $F = 1$. Thus:

$$U = \frac{1 + G^2}{1 + G}.$$

- Minimizing U over G we have:

$$\frac{dU}{dG} = \frac{2G(1+G) - (1+G^2)}{(1+G)^2} = \frac{G^2 + 2G - 1}{(1+G)^2},$$

- And, $dU/dG = 0$ for $G^2 + 2G - 1 = 0$, which has two solutions:

$$\begin{cases} G_1 = -1 - \sqrt{2} \\ G_2 = -1 + \sqrt{2}. \end{cases}$$

Least Upper Bound (U_{LUB}) of RM (8)

- Since $0 \leq \mathrm{G} < 1$, the negative solution $G = G_1$ is discarded. Thus, the least upper bound of U is given for $G = G_2$:

$$U_{lub} = \frac{1 + (\sqrt{2} - 1)^2}{1 + (\sqrt{2} - 1)} = \frac{4 - 2\sqrt{2}}{\sqrt{2}} = 2(\sqrt{2} - 1).$$

$$U_{lub} = 2(2^{1/2} - 1) \simeq 0.83.$$

- Notice that, if T_2 is a multiple of T_1, $G = 0$ and the processor utilization factor becomes 1.
- In general, the utilization factor for two tasks can be computed as a function of the ratio $k = T_2/T_1$. For a given F, from slide 27, we can write:

$$U = \frac{F + (k - F)^2}{k} = k - 2F + \frac{F(F + 1)}{k}.$$

Least Upper Bound (U_{LUB}) of RM (9)

- Minimizing U over k we have:

$$\frac{dU}{dk} = 1 - \frac{F(F+1)}{k^2},$$

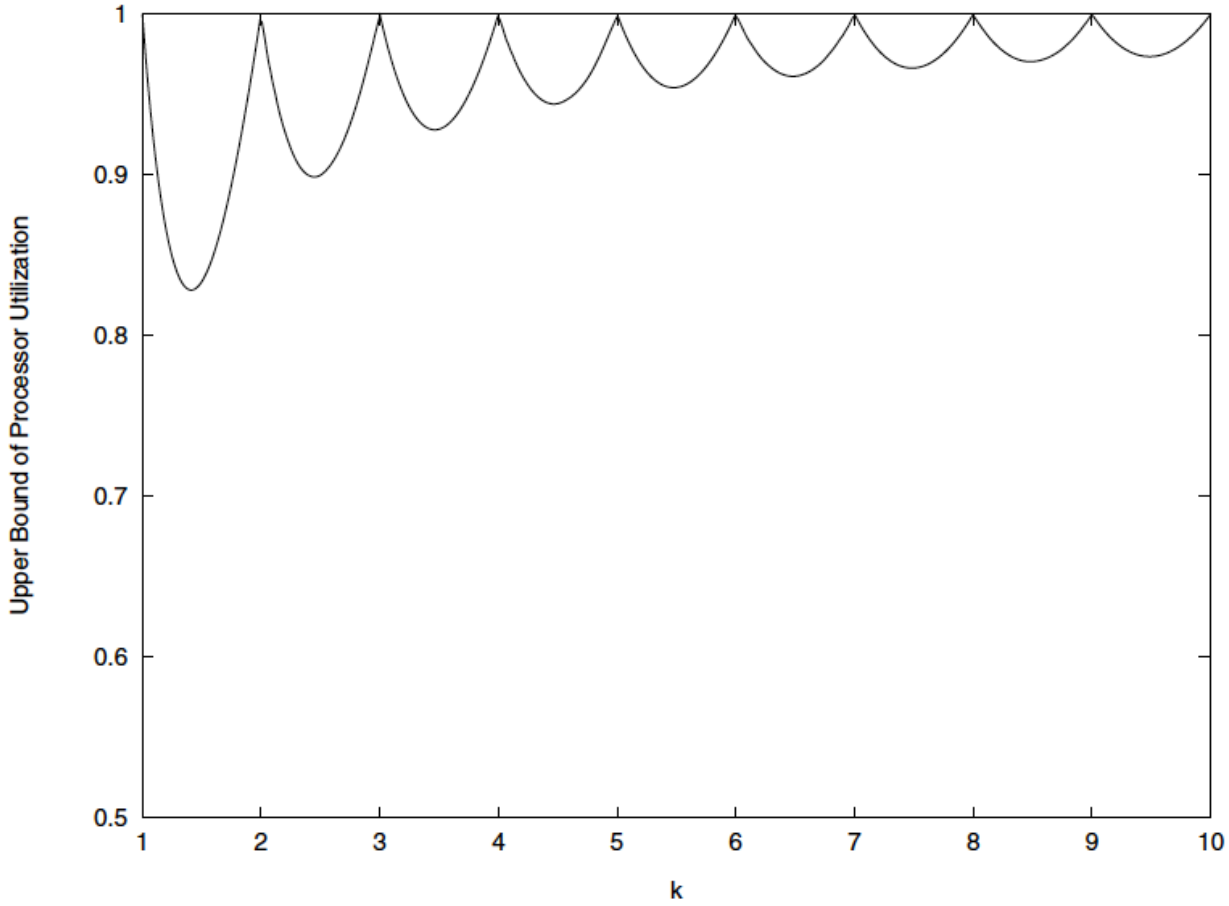
- And, $dU/dk=0$ for $k^* = \sqrt{F(F+1)}$. Hence, for a given F, the minimum value of U is:

$$U^* = 2(\sqrt{F(F+1)} - F).$$

F	k^*	U^*
1	$\sqrt{2}$	0.828
2	$\sqrt{6}$	0.899
3	$\sqrt{12}$	0.928
4	$\sqrt{20}$	0.944
5	$\sqrt{30}$	0.954

Least Upper Bound (U_{LUB}) of RM (10)

- Upper bound of the processor utilization factor as a function of the ratio $k = T_2/T_1$.



Calculation of U_{LUB} for n Tasks

- The conditions that allow to compute the least upper bound of the processor utilization factor are:

$$\begin{cases} F = 1 \\ C_1 = T_2 - FT_1 \\ C_2 = (T_1 - C_1)F, \end{cases} \longrightarrow \begin{cases} T_1 < T_2 < 2T_1 \\ C_1 = T_2 - T_1 \\ C_2 = 2T_1 - T_2. \end{cases}$$

- Generalizing for an arbitrary set of n tasks, the worst conditions for the schedulability of a task set that fully utilizes the processor are:

$$\begin{cases} T_1 < T_n < 2T_1 \\ C_1 = T_2 - T_1 \\ C_2 = T_3 - T_2 \\ \dots \\ C_{n-1} = T_n - T_{n-1} \\ C_n = T_1 - (C_1 + C_2 + \dots + C_{n-1}) = 2T_1 - T_n. \end{cases}$$

Calculation of U_{LUB} for n Tasks (2)

- Thus, the processor utilization factor becomes:

$$U = \frac{T_2 - T_1}{T_1} + \frac{T_3 - T_2}{T_2} + \ldots + \frac{T_n - T_{n-1}}{T_{n-1}} + \frac{2T_1 - T_n}{T_n}.$$

- Defining:

$$R_i = \frac{T_{i+1}}{T_i}$$

- Noting that:

$$R_1 R_2 \ldots R_{n-1} = \frac{T_n}{T_1},$$

- The utilization factor may be written as:

$$U = \sum_{i=1}^{n-1} R_i + \frac{2}{R_1 R_2 \ldots R_{n-1}} - n.$$

- To minimize U over R_i, $i = 1, \ldots, n-1$, we have:

$$\frac{\partial U}{\partial R_k} = 1 - \frac{2}{R_i^2 (\prod_{i \neq k}^{n-1} R_i)}.$$

Calculation of U_{LUB} for n Tasks (3)

- Thus, defining $P = R_1 R_2 \ldots R_{n-1}$, U is minimum when:

$$\begin{cases} R_1 P = 2 \\ R_2 P = 2 \\ \ldots \\ R_{n-1} P = 2. \end{cases}$$

- That is, when all R_i have the same value:

$$R_1 = R_2 = \ldots = R_{n-1} = 2^{1/n}.$$

- Substituting this value in U we obtain:

$$\begin{aligned} U_{lub} &= (n-1)2^{1/n} + \frac{2}{2^{(1-1/n)}} - n = \\ &= n2^{1/n} - 2^{1/n} + 2^{1/n} - n = \\ &= n(2^{1/n} - 1). \end{aligned}$$

Calculation of U_{LUB} for n Tasks (4)

- Therefore, for an arbitrary set of periodic tasks, the least upper bound of the processor utilization factor under the Rate Monotonic scheduling algorithm is:

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

- For high values of n, the least upper bound converges to:

$$U_{lub} = \ln 2 \simeq 0.69.$$

Earliest Deadline First (EDF) Scheduling

- A dynamic scheduling rule that selects tasks according to their absolute deadlines.
- Tasks with earlier deadlines will be executed at higher priorities.
- Since the absolute deadline of a periodic task depends on the current j^{th} instance as:

$$d_{i,j} = \Phi_i + (j-1)T_i + D_i,$$

- Typically executed in preemptive mode, thus the currently executing task is preempted whenever another periodic instance with earlier deadline becomes active.

Schedulability Analysis of EDF Scheduling

- Under the assumptions A1, A2, A3, and A4, the schedulability of a periodic task set handled by EDF can be verified through the processor utilization factor.
- The least upper bound is one; therefore, tasks may utilize the processor up to 100% and still be schedulable.
- **Theorem.** A set of periodic tasks is schedulable with EDF if and only if:

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

Schedulability Analysis of EDF Scheduling

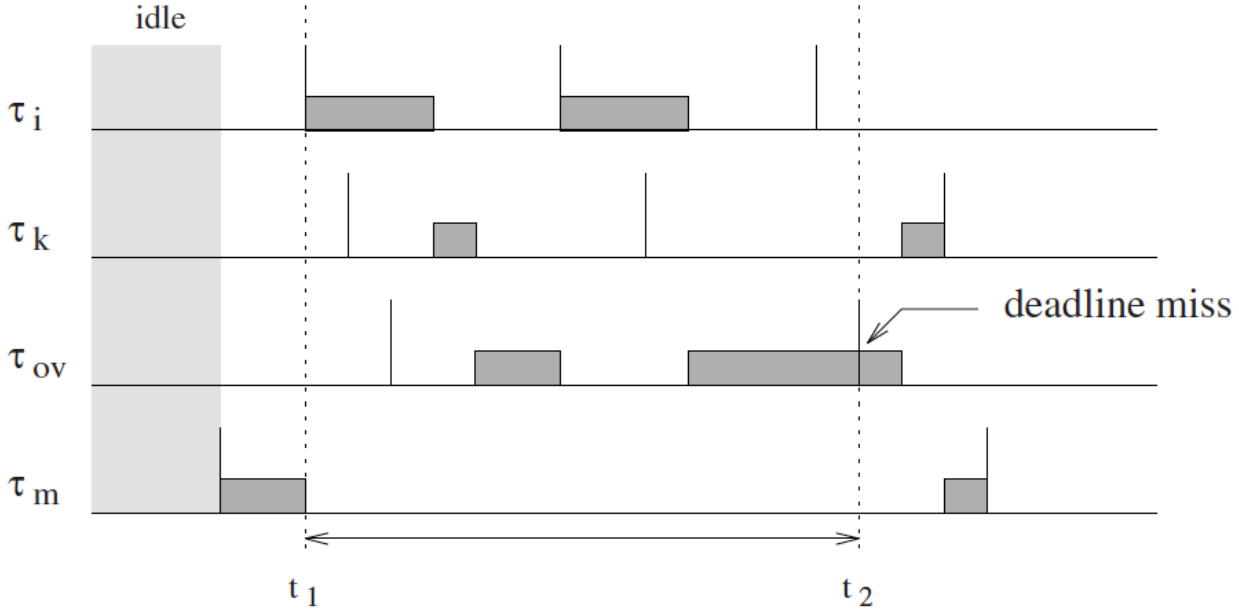
- **Proof.** *Only if*. We show that a task set cannot be scheduled if $U>1$.
- In fact, by defining $T=T_1T_2 \ldots T_n$, the total demand of computation time requested by all tasks in T can be calculated as:

$$\sum_{i=1}^{n} \frac{T}{T_i} C_i = UT.$$

- If $U>1$, that is, if the total demand UT exceeds the available processor time T, there is clearly no feasible schedule for the task set.

Schedulability Analysis of EDF Scheduling (2)

- *If.* Assume that the condition $U<1$ is satisfied and yet the task set is not schedulable.



- Let $C_p(t_1, t_2)$ be the total computation time demanded by periodic tasks in $[t_1, t_2]$, which can be computed as:

$$C_p(t_1, t_2) = \sum_{r_k \geq t_1, d_k \leq t_2} C_k = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i.$$

Schedulability Analysis of EDF Scheduling (3)

- Observe that:

$$C_p(t_1, t_2) = \sum_{i=1}^{n} \left\lfloor \frac{t_2 - t_1}{T_i} \right\rfloor C_i \leq \sum_{i=1}^{n} \frac{t_2 - t_1}{T_i} C_i = (t_2 - t_1)U.$$

- Since a deadline is missed at t_2, $C_p(t_1, t_2)$ must be greater than the available processor time $(t_2 - t_1)$; thus, we must have:

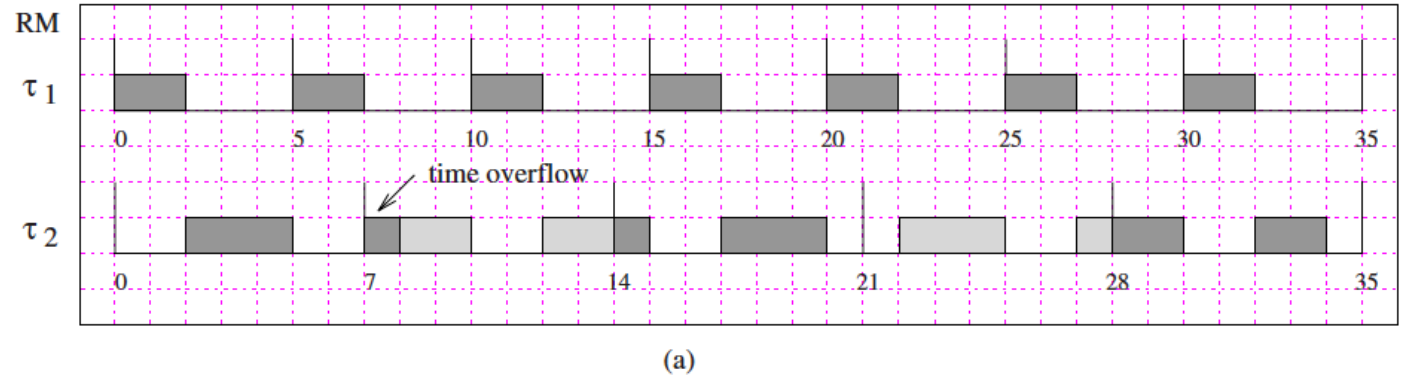
$$(t_2 - t_1) < C_p(t_1, t_2) \leq (t_2 - t_1)U.$$

- That is, $U>1$, which is a contradiction.

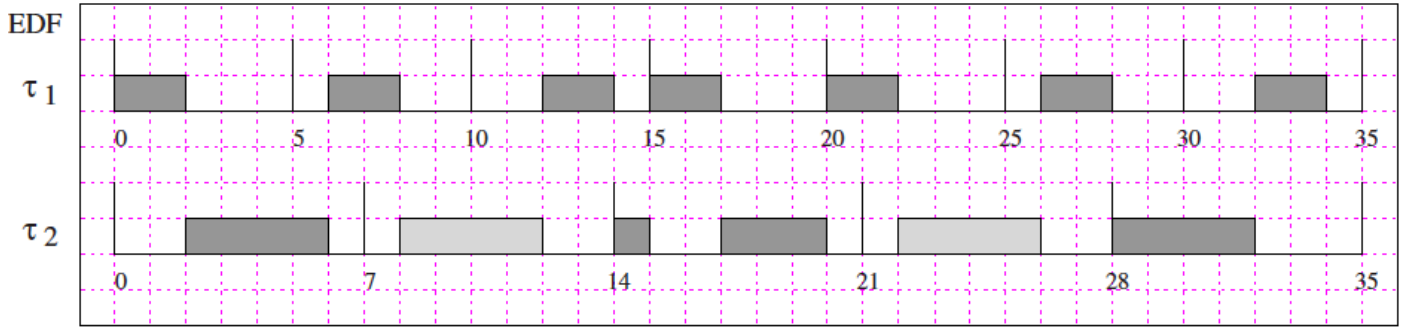
An Example of difference between RM and EDF

- Consider the periodic task set, for which the processor utilization factor is:

$$U = \frac{2}{5} + \frac{4}{7} = \frac{34}{35} \simeq 0.97.$$



(a)



(b)

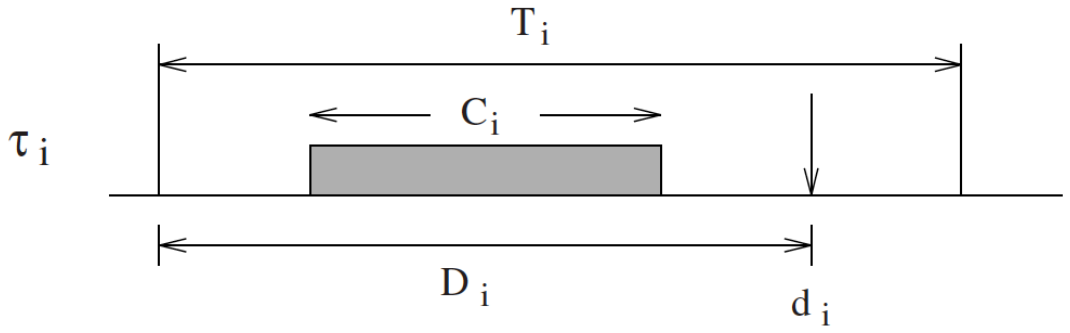
Deadline Monotonic Scheduling

- The algorithms and the schedulability bounds illustrated in the previous slides rely on the assumptions A1, A2, A3, and A4 presented at the beginning of this lecture.
- In particular, assumption A3 imposes a relative deadline equal to the period, allowing an instance to be executed anywhere within its period.
  - This condition could not always be desired in real-time applications.
- The Deadline Monotonic (DM) priority assignment weakens the “period equals deadline” constraint within a static priority scheduling scheme.
- This algorithm was first proposed in 1982 by Leung and Whitehead [LW82] as an extension of Rate Monotonic.

Deadline Monotonic Scheduling (2)

- Tasks can have relative deadlines less than or equal to their period (i.e., constrained deadlines).
- Each periodic task τ_i is characterized by four parameters:
  - A phase Φ_i;
  - A worst-case computation time C_i (constant for each instance);
  - A relative deadline D_i (constant for each instance);
  - A period T_i.

$$\begin{cases} C_i \leq D_i \leq T_i \\ r_{i,k} = \Phi_i + (k-1)T_i \\ d_{i,k} = r_{i,k} + D_i. \end{cases}$$



Deadline Monotonic Scheduling (3)

- Each task is assigned a fixed priority P_i inversely proportional to its relative deadline D_i.
- At any instant, the task with the shortest relative deadline is executed.
- Since relative deadlines are constant, DM is a static priority assignment.
- As RM, DM is normally used in a fully preemptive mode.
  - The currently executing task is preempted by a newly arrived task with shorter relative deadline.
- The Deadline-Monotonic priority assignment is optimal, meaning that, if a task set is schedulable by some fixed priority assignment, then it is also schedulable by DM.

Schedulability Analysis of DM

- The feasibility of a task set with constrained deadlines could be guaranteed using the utilization based test, by reducing tasks' periods to relative deadlines:

$$\sum_{i=1}^{n} \frac{C_i}{D_i} \leq n(2^{1/n} - 1).$$

  - Such a test would be quite pessimistic, since the workload on the processor would be overestimated.

- A less pessimistic schedulability test can be derived by noting that:
  - The worst-case processor demand occurs when all tasks are released simultaneously; that is, at their critical instants;
  - For each task τ_i, the sum of its processing time and the interference (preemption) imposed by higher priority tasks must be less than or equal to D_i.

Schedulability Analysis of DM (2)

- Assuming that tasks are ordered by increasing relative deadlines, so that:

$$i < j \iff D_i < D_j,$$

- Such a test can be expressed as follows:

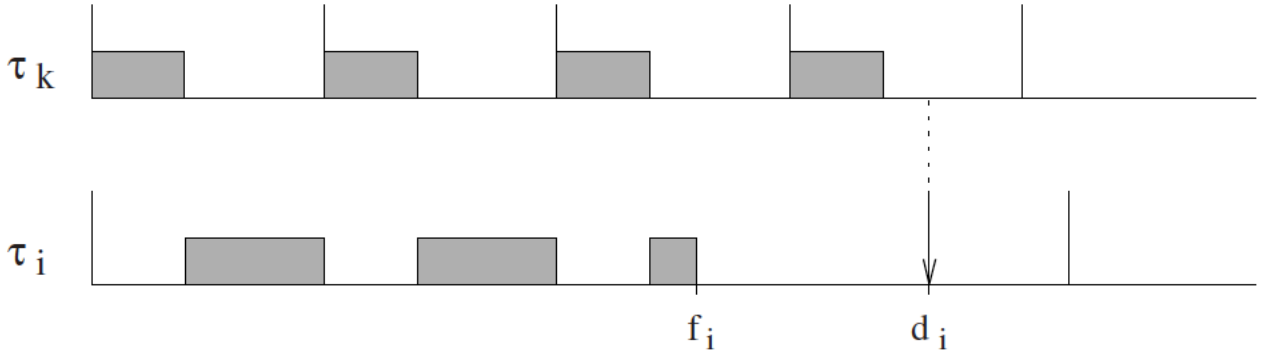
$$\forall i : 1 \leq i \leq n \quad C_i + I_i \leq D_i,$$

- where I_i is a measure of the interference on τ_i, which can be computed as the sum of the processing times of all higher-priority tasks released before D_i:

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{D_i}{T_j} \right\rceil C_j.$$

Schedulability Analysis of DM (3)

- This test is sufficient but not necessary for guaranteeing the schedulability of the task set.
  - This is due to the fact that I_i is calculated by assuming that each higher priority task τ_j exactly interferes $\left\lceil \frac{D_i}{T_j} \right\rceil$ times on τ_i.
  - The actual interference can be smaller than I_i, since τ_i may terminate earlier.



- Evaluating the exact interference on periodic tasks and deriving a sufficient and necessary schedulability test for DM, called ***Response Time Analysis***.

Response Time Analysis of DM

- The longest response time R_i of a periodic task τ_i is computed, at the critical instant, as the sum of its computation time and the interference I_i of the higher priority tasks:

$$R_i = C_i + I_i,$$

$$I_i = \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j.$$

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j.$$

- No simple solution exists for this equation since R_i appears on both sides.
- The worst-case response time of task τ_i is given by the smallest value of R_i that satisfies the last equation.

Response Time Analysis of DM (2)

- Only a subset of points in the interval $[0, D_i]$ need to be examined for feasibility.
  - In fact, the interference on τ_i only increases when there is a release of a higher-priority task.
- To simplify the notation, let $R_i^{(k)}$ be the k-th estimate of R_i and let $I_i^{(k)}$ be the interference on task τ_i in the interval $[0, R_i^{(k)}]$:

$$I_i^{(k)} = \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j .$$

Response Time Analysis of DM (3)

- The calculation of R_i is performed as follows:
  - 1. Iteration starts with $R_i^{(0)} = \sum_{j=1}^{i} C_j$, which is the first point in time that τ_i could possibly complete.
  - 2. The actual interference $I_i^{(k)}$ in the interval $[0, R_i^{(k)}]$ is computed by the equation in the previous slide.

$$I_i^{(k)} = \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j.$$

  - 3. If $I_i^{(k)} + C_i = R_i^{(k)}$, then $R_i^{(k)}$ is the actual worst-case response time of task τ_i; that is, $R_i = R_i^{(k)}$. Otherwise, the next estimate is given by:

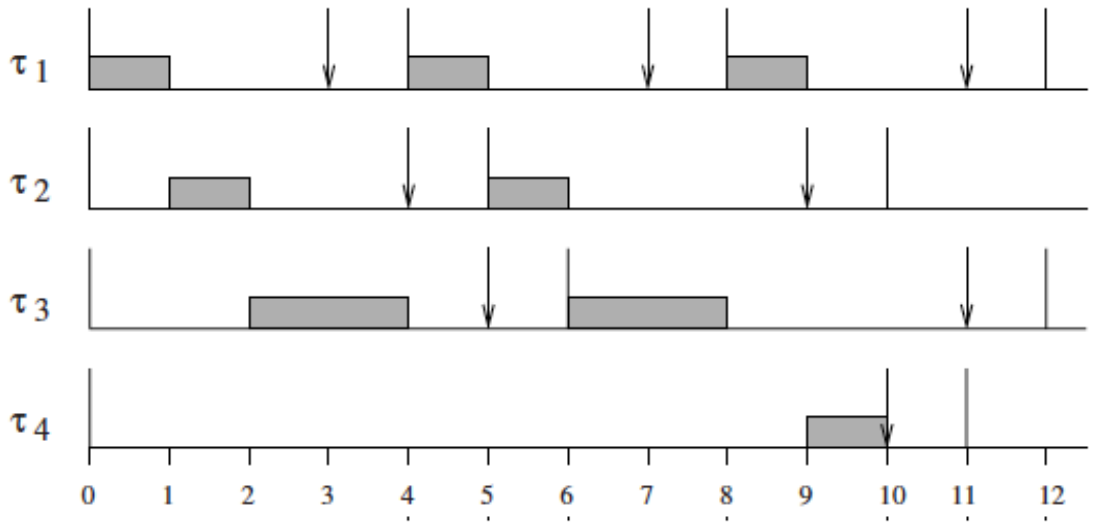
$$R_i^{(k+1)} = I_i^{(k)} + C_i,$$

  - and the iteration continues from step 2.
- Once R_i is calculated, the feasibility of task τ_i is guaranteed if and only if $R_i \leq D_i$.

An Example of Response Time Analysis of DM

- To clarify the schedulability test, consider the set of periodic tasks shown in the following table, simultaneously activated at time $t=0$.
- In order to guarantee τ_4, we have to calculate R_4 and verify that $R_4 \leq D_4$.

	C_i	T_i	D_i
τ_1	1	4	3
τ_2	1	5	4
τ_3	2	6	5
τ_4	1	11	10



An Example of Response Time Analysis of DM (2)

	C_i	T_i	D_i
τ_1	1	4	3
τ_2	1	5	4
τ_3	2	6	5
τ_4	1	11	10

$$I_i^{(k)} = \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j.$$

$$R_i^{(k+1)} = I_i^{(k)} + C_i,$$

Step 0: $R_4^{(0)} = \sum_{i=1}^{4} C_i = 5$, but $I_4^{(0)} = 5$ and $I_4^{(0)} + C_4 > R_4^{(0)}$ hence τ_4 does not finish at $R_4^{(0)}$.

Step 1: $R_4^{(1)} = I_4^{(0)} + C_4 = 6$, but $I_4^{(1)} = 6$ and $I_4^{(1)} + C_4 > R_4^{(1)}$ hence τ_4 does not finish at $R_4^{(1)}$.

Step 2: $R_4^{(2)} = I_4^{(1)} + C_4 = 7$, but $I_4^{(2)} = 8$ and $I_4^{(2)} + C_4 > R_4^{(2)}$ hence τ_4 does not finish at $R_4^{(2)}$.

Step 3: $R_4^{(3)} = I_4^{(2)} + C_4 = 9$, but $I_4^{(3)} = 9$ and $I_4^{(3)} + C_4 > R_4^{(3)}$ hence τ_4 does not finish at $R_4^{(3)}$.

Step 4: $R_4^{(4)} = I_4^{(3)} + C_4 = 10$, but $I_4^{(4)} = 9$ and $I_4^{(4)} + C_4 = R_4^{(4)}$ hence τ_4 finishes at $R_4 = R_4^{(4)} = 10$.

- Since $R_4 \leq D_4$, τ_4 is schedulable within its deadline.

Schedulability Test Algorithm for DM

- If $R_i \leq D_i$ for all tasks, we conclude that the task set is schedulable by DM. Such a schedulability test can be performed by the algorithm illustrated in the following:

```
DM_guarantee (Γ) {
    for (each τ_i ∈ Γ) {
        I_i = Σ_{k=1}^{i-1} C_k;
        do {
            R_i = I_i + C_i;
            if (R_i > D_i) return(UNSCHEDULABLE);
            I_i = Σ_{k=1}^{i-1} ⌈R_i / T_k⌉ C_k;
        } while (I_i + C_i > R_i);
    }
    return(SCHEDULABLE);
}
```

EDF with Constrained Deadlines

- Under EDF, the analysis of periodic tasks with deadlines less than or equal to periods can be performed using the *processor demand* criterion.
- In general, the processor demand of a task τ_i in an interval $[t_1, t_2]$ is the amount of processing time $g_i(t_1, t_2)$ requested by those instances of τ_i activated in $[t_1, t_2]$ that must be completed in $[t_1, t_2]$. That is:

$$g_i(t_1, t_2) = \sum_{r_{i,k} \geq t_1, d_{i,k} \leq t_2} C_i.$$

- For the whole task set, the processor demand in $[t_1, t_2]$ is given by:

$$g(t_1, t_2) = \sum_{i=1}^{n} g_i(t_1, t_2).$$

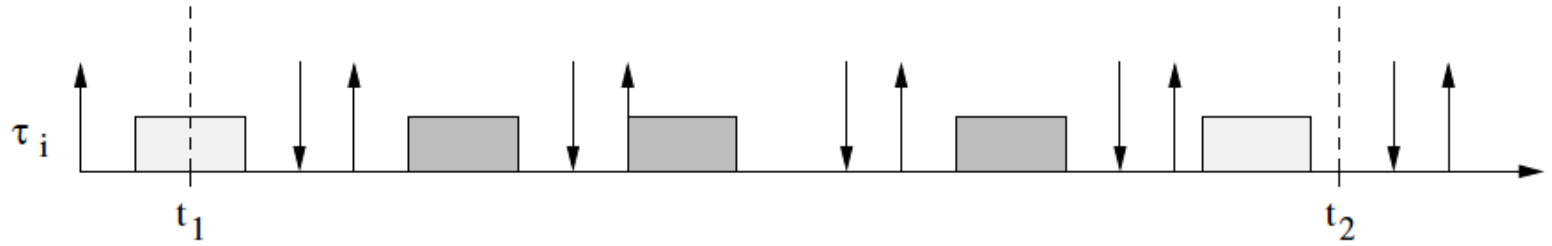
EDF with Constrained Deadlines (2)

- The feasibility of a task set is guaranteed if and only if in *any interval of time* the processor demand does not exceed the available time; that is, if and only if:

$$\forall t_1, t_2 \quad g(t_1, t_2) \leq (t_2 - t_1).$$

- The number of instances of task τ_i that contribute to the demand in $[t_1, t_2]$ can be expressed as:

$$\eta_i(t_1, t_2) = max\left\{0, \left\lfloor \frac{t_2 + T_i - D_i - \Phi_i}{T_i} \right\rfloor - \left\lceil \frac{t_1 - \Phi_i}{T_i} \right\rceil \right\}$$



EDF with Constrained Deadlines (3)

- and the processor demand in $[t_1, t_2]$ can be computed as:

$$g(t_1, t_2) = \sum_{i=1}^{n} \eta_i(t_1, t_2) C_i.$$

- If relatives deadlines are no larger than periods and periodic tasks are simultaneously activated at time t=0 (i.e., Φ_i=0 for all the tasks), then the number of instances contributing to the demand in an interval $[0, L]$ can be expressed as:

$$\eta_i(0, L) = \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor.$$

- The processor demand in $[0, L]$ can be computed as:

$$g(0, L) = \sum_{i=1}^{n} \left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor C_i.$$

Demand Bound Function for EDF

- Function $g(0, L)$ is also referred to as **Demand Bound Function**:

$$\mathsf{dbf}(t) \stackrel{\text{def}}{=} \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i.$$

- Therefore, a synchronous set of periodic tasks with relative deadlines less than or equal to periods is schedulable by EDF if and only if:

$$\forall t > 0 \quad \mathsf{dbf}(t) \leq t.$$

DBF of Periodic Tasks with $D_i=T_i$

- **Theorem.** *A set of periodic tasks with relative deadlines equal to periods is schedulable by EDF if and only if*

$$\forall L > 0 \quad \sum_{i=1}^{n} \left\lfloor \frac{L}{T_i} \right\rfloor C_i \leq L.$$

Reducing Test Intervals

1. If tasks are periodic and are simultaneously activated at time $t = 0$, then the schedule repeats itself every hyperperiod H; thus the schedulability condition needs to be verified only for values of L less than or equal to H.
2. $g(0, L)$ is a step function whose value increases when L crosses a deadline d_k and remains constant until the next deadline d_{k+1}. This means that if condition $g(0, L) < L$ holds for $L = d_k$, then it also holds for all L such that $d_k \leq L < d_{k+1}$. As a consequence, the schedulability condition needs to be verified only for values of L equal to absolute deadlines.

Reducing Test Intervals

The number of testing points can be reduced further by noting that

$$\left\lfloor \frac{L + T_i - D_i}{T_i} \right\rfloor \leq \left(\frac{L + T_i - D_i}{T_i} \right).$$

and defining

$$G(0, L) = \sum_{i=1}^{n} \frac{L + T_i - D_i}{T_i} C_i = \sum_{i=1}^{n} \frac{T_i - D_i}{T_i} C_i + \frac{L}{T_i} C_i$$

we have that

$$\forall L > 0, \quad g(0, L) \leq G(0, L),$$

where

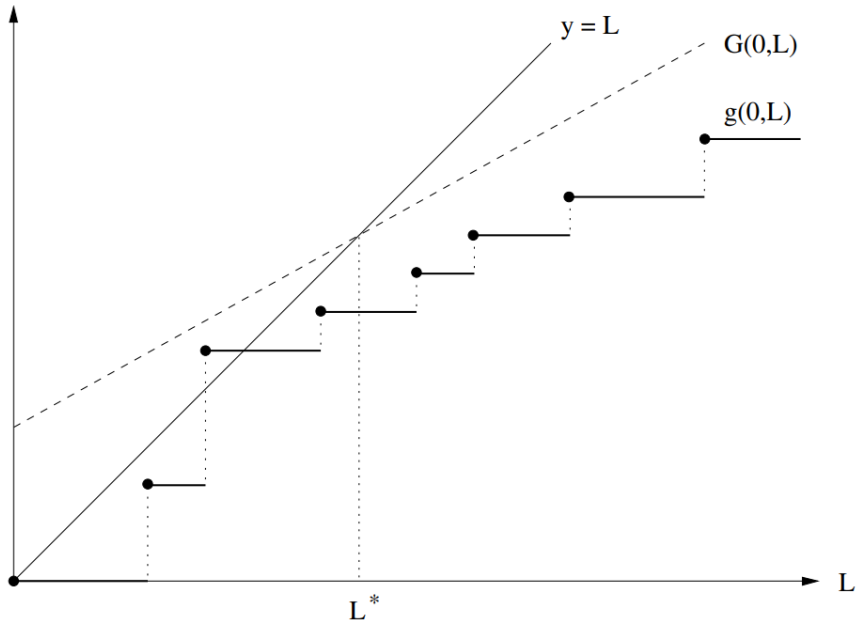
$$G(0, L) = \sum_{i=1}^{n} (T_i - D_i) U_i + LU.$$

Reducing Test Intervals

- From figure, we can note that $G(0, L)$ is a function of L increasing as a straight line with slope U.
- Hence, if $U < 1$, there exists an $L = L*$ for which $G(0, L) = L$. Clearly, for all $L \geq L*$, we have that $g(0, L) \leq G(0, L) \leq L$, meaning that the schedulability of the task set is guaranteed. As a consequence, there is no need to verify schedulability condition for values of $L \geq L*$.

$$\sum_{i=1}^{n}(T_i - D_i)U_i + L^*U = L^*,$$

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1 - U}.$$



Reducing Test Intervals

- **Theorem.** *A set of synchronous periodic tasks with relative deadlines less than or equal to periods can be scheduled by EDF if and only if* $U<1$ *and*

$$\forall t \in \mathcal{D} \quad \mathsf{dbf}(t) \leq t.$$

where

$$\mathcal{D} = \{d_k \mid d_k \leq \min[H, \max(D_{max}, L^*)]\}$$

and

$$L^* = \frac{\sum_{i=1}^{n}(T_i - D_i)U_i}{1-U}.$$

Example

- To illustrate the processor demand criterion, consider three periodic tasks with deadlines less than periods need to be guaranteed under EDF.

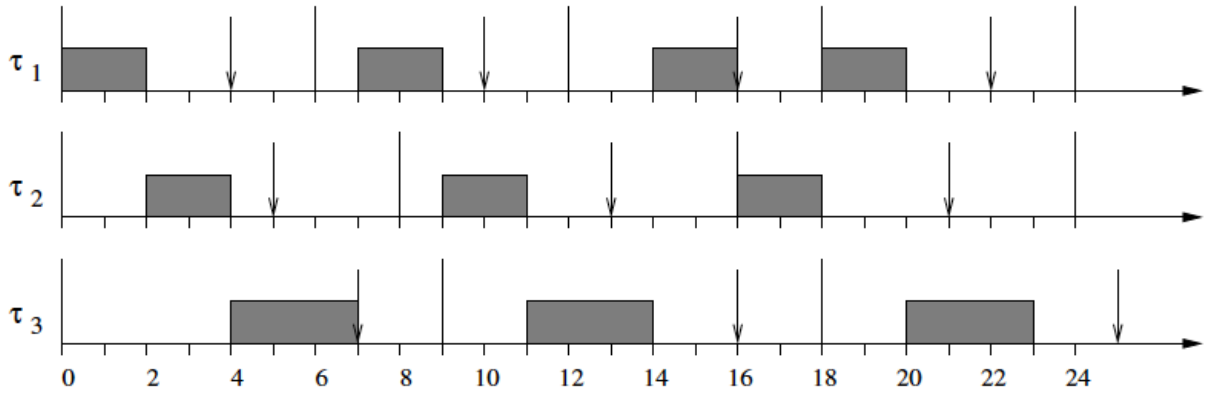
	C_i	D_i	T_i
τ_1	2	4	6
τ_2	2	5	8
τ_3	3	7	9

$$
\begin{aligned}
U &= \frac{2}{6} + \frac{2}{8} + \frac{3}{9} = \frac{11}{12} \\
L^* &= \frac{\sum_{i=1}^{n} (T_i - D_i) U_i}{1 - U} = 25 \\
H &= \text{lcm}(6, 8, 9) = 72.
\end{aligned}
$$

$$\mathcal{D} = \{4, 5, 7, 10, 13, 16, 21, 22\}$$

$$
\begin{aligned}
r_{i,k} &= \Phi_i + (k-1)T_i \\
d_{i,k} &= r_{i,k} + D_i.
\end{aligned}
$$

L	$g(0, L)$	result
4	2	OK
5	4	OK
7	7	OK
10	9	OK
13	11	OK
16	16	OK
21	18	OK
22	20	OK



Summary

- Parameters of Periodic Tasks
- Utilization Factor
- Time-Line Scheduling
- Rate Monotonic Scheduling
- Earliest Deadline First Scheduling
- Deadline Monotonic Scheduling