Periodic Tasks Scheduling Algorithms (Chapter 4)

Sections covered:

1. Processor utilization factor
2. Timeline scheduling
3. Rate monotonic scheduling
4. Earliest deadline first scheduling
5. Deadline monotonic
6. EDF with constrained deadlines

Periodic Task Scheduling

- In many real-time control applications, periodic activities represent the major computational demand in the system.
- Periodic tasks typically arise from sensory data acquisition, low-level servoing, control loops, action planning, and system monitoring (See Chapter 11 for examples)
- When a control application consists of several concurrent periodic tasks with individual timing constraints, the operating system must guarantee that each periodic instance is regularly activated at its proper rate and is completed within its deadline
- Algorithms presented are: timeline scheduling, rate monotonic, earliest deadline first, and deadline monotonic

Notations

- Γ denotes a set of periodic tasks;
- τ_i denotes a generic periodic task;
- $\tau_{i,j}$ denotes the j^{th} instance of task τ_i;
- $r_{i,j}$ denotes the release time of the j^{th} 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 j^{th} 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 j^{th} instance of task τ_i; that is, the time at which it starts executing.
- $f_{i,j}$ denotes the finishing time of the j^{th} instance of task τ_i; that is, the time at which it completes the execution.

Schedulability Analysis, Hypotheses Assumed on Tasks

- **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

Periodic Task Parameters

- In the cases, in which the assumptions A1, A2, A3, and A4 hold, a periodic task τ_i can be completely characterized by the following three parameters: its phase Φ_i, its period T_i and its worst-case computation time C_i.
- Thus, a set of periodic tasks can be denoted by:

$$\Gamma = \{\tau_i(\Phi_i, T_i, C_i), i = 1, \ldots, n\}.$$

- The release time $r_{i,k}$ and the absolute deadline $d_{i,k}$ of the generic k^{th} instance can then be computed as:

$$r_{i,k} = \Phi_i + (k - 1)T_i$$

$$d_{i,k} = r_{i,k} + T_i = \Phi_i + kT_i$$

Periodic Task Parameters

- **Hyperperiod:** the minimum interval of time H after which the schedule repeats itself; the schedule in [0,H] is the same as that in [kH, (k + 1)H] for any integer k > 0.
  - For a set of periodic tasks synchronously activated at time t = 0: H = lcm(T_1, . . . , T_n). [lcm stands for least common multiple]
- **Job response time:** the time (measured from the release time) at which the job is terminated: $R_{i,k} = f_{i,k} - r_{i,k}$.
- **Task response time:** the maximum response time among all the jobs:
  $R_i = \max_k R_{i,k}$; for all k
- **Critical instant of a task:** the arrival time that produces 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

Periodic Task Parameters

- **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.** It is the maximum deviation of the start time among all instances:
  $ARJ_i = \max_k (s_{i,k} - r_{i,k}) - \min_k (s_{i,k} - r_{i,k})$.
- **Relative Finishing Jitter of a task.** It is 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.** It is the maximum deviation of the finishing time among all instances:
  $AFJ_i = \max_k (f_{i,k} - r_{i,k}) - \min_k (f_{i,k} - r_{i,k})$

Feasibility

- A periodic task τ_i is said to be feasible if all its instances finish within their deadlines.
- A task set Γ is said to be schedulable (or feasible) if all tasks in Γ are feasible.

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.
- The CPU utilization can be improved by increasing tasks' computation times or by decreasing their periods,
  - However, there exists a maximum value of U below which Γ is schedulable and above which Γ is not schedulable

Upper Bound Processor Utilization Factor

- Let $U_{ub}(\Gamma,A)$ be the upper bound of the processor utilization factor for a task set Γ under a given algorithm A.
- When $U = U_{ub}(\Gamma,A)$, the set Γ is said to fully utilize the processor:
  - In this situation, Γ is schedulable by A, but an increase in the computation time in any of the tasks will make the set infeasible.

Example of Upper Bound Processor Utilization Factor

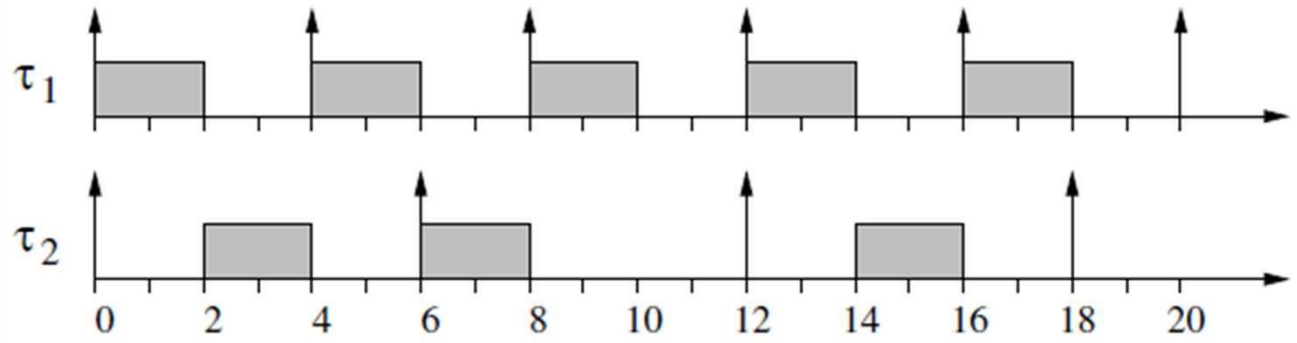


Figure 4.1 A task set with $U_{ub} = 5/6$.

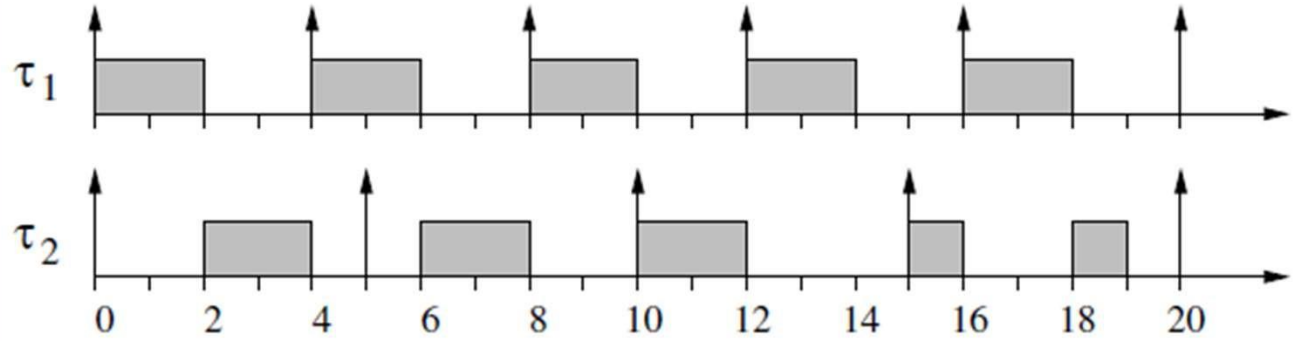


Figure 4.2 A task set with $U_{ub} = 0.9$.

The Least Upper Bound $U_{lub}(A)$ of an Algorithm A

- $U_{lub}(A) = \min_{\Gamma} U_{ub}(\Gamma, A)$
- $U_{ub}(A)$ -- each task set Γ_i fully utilizes the processor when its utilization factor U_i (varied by changing tasks' computation times) reaches a particular upper bound U_{ubi}.
  - If $U_i \leq U_{ubi}$, then Γ_i is schedulable, else Γ_i is not schedulable.
- As a result, any task set having a processor utilization factor U below $U_{lub}(A)$ is certainly schedulable by A.

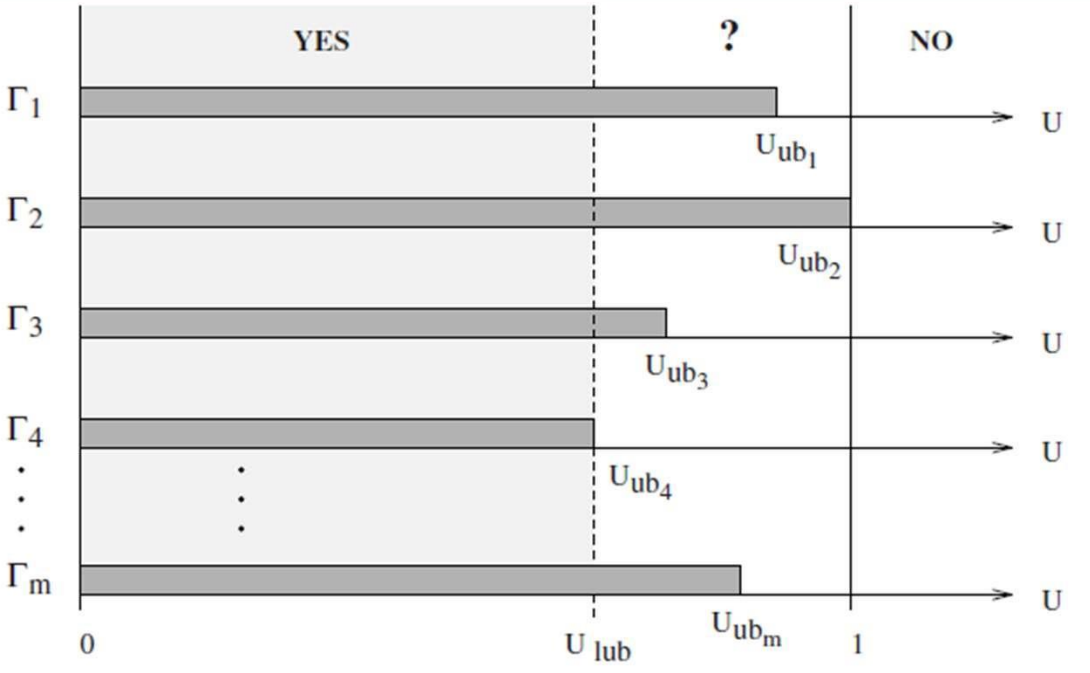


Figure 4.3 Meaning of the least upper bound of the processor utilization factor.

The task sets Γi shown in the figure differ for the number of tasks and for the configuration of their periods.

U_{lub} Considerations

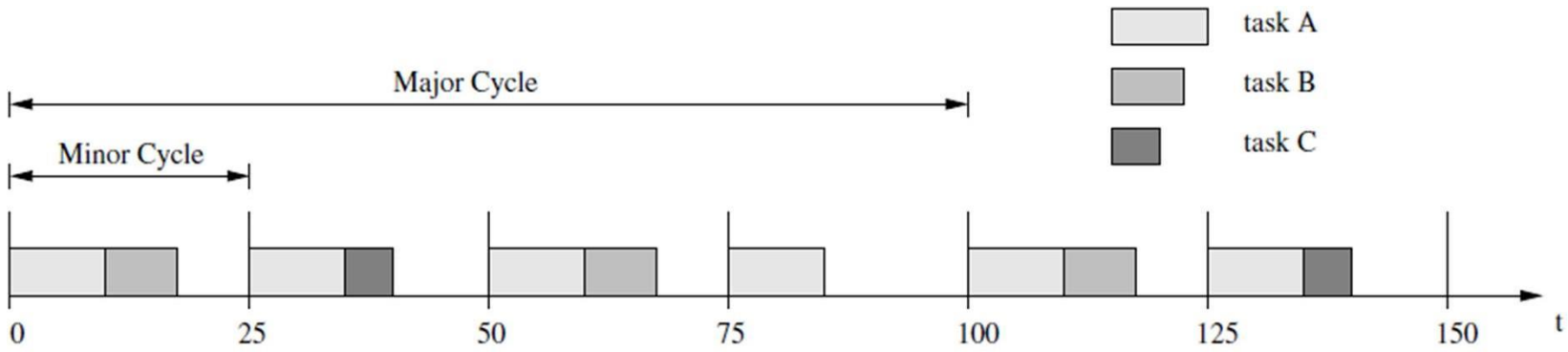
- U_{lub} is useful for easily verifying the schedulability of a task set
- Any task set whose processor utilization factor is less than or equal to this bound is schedulable by the algorithm
- When $U_{lub} < U \leq 1$ the schedulability can be achieved only if the task periods are suitably related

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

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

Timeline Scheduling (TS) or Cyclic Executive

- TS is one of the most used algorithms to handle periodic tasks in defense military systems and traffic control systems.
- Methodology: divide the temporal axis into slots of equal length
  - Slot length should be equal to the greatest common divisor of the periods
- A timer synchronizes the activation of the tasks at the beginning of each time slot
- Example … in class …



Minor and Major Cycles

- The duration of the time slot is also called a Minor Cycle, and is equal to the greatest common divisor of the periods
- The minimum interval of time after which the schedule repeats itself (the hyperperiod H) is also called a Major Cycle
  - In general, the major cycle is equal to the least common multiple of all the periods

TS Feasibility

- In order to guarantee a priori that a schedule is feasible on a particular processor, it is sufficient to know the task worst-case execution times and verify that the sum of the executions within each time slot is less than or equal to the minor cycle.

- For the example we must show that:

$$\begin{cases} C_A + C_B \leq 25ms \\ C_A + C_C \leq 25ms \end{cases}$$

Implementation, Advantages, Disadvantages

- TS Implementation. 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
- Advantages … simplicity… low overhead … no task jitter …
- Disadvantages … fragile during overload conditions … sensitive to application changes … difficulty to handle aperiodic tasks …

Rate Monotonic Scheduling

- The Rate Monotonic (RM) scheduling algorithm is based on a simple rule that assigns priorities to tasks according to their request rates.
- Tasks with higher request rates (that is, with shorter periods) will have higher priorities.
- Since periods are constant, RM is a fixed-priority assignment:
  - A priority *Pi* is assigned to the task before execution and does not change over time.
- Moreover, RM is intrinsically preemptive: the currently executing task is preempted by a newly arrived task with shorter period.
- Liu and Layland
  - showed that RM is optimal among all fixed-priority assignments
  - derived the U_{lub} factor for a generic set of n periodic tasks.

Optimality of RM

- First show that: a critical instant for any task occurs whenever the task is released simultaneously with all higher priority tasks.
- Let Γ = {τ1, τ2, . . . , τn} be a set of periodic tasks ordered by increasing periods => τn has the lowest priority.
- The worst response time of a task occurs when it is released simultaneously with all higher-priority tasks
  - Proof … analyze graphs a and b … ⇨

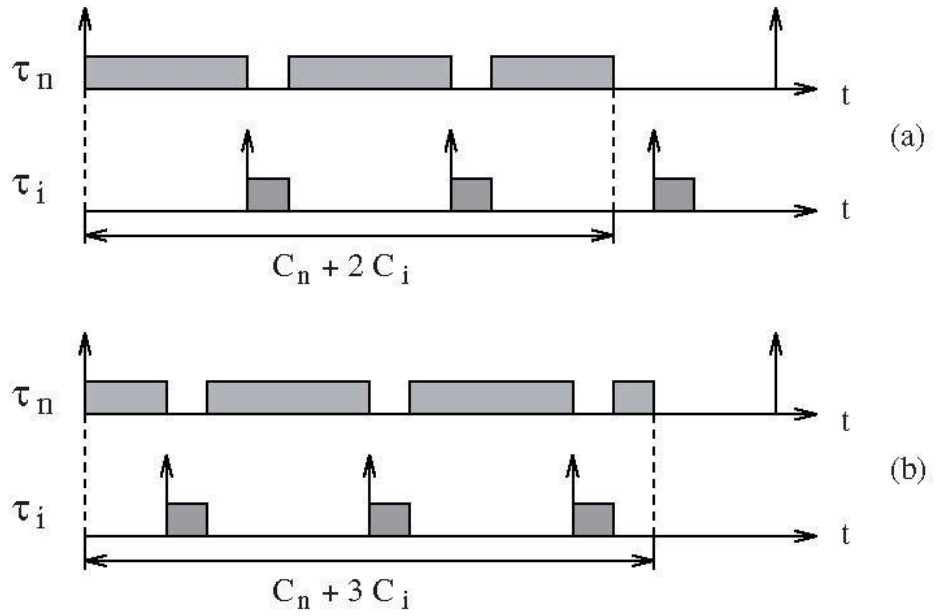


Figure 4.5 **a.** The response time of task τ_n is delayed by the interference of τ_i with higher priority. **b.** The interference may increase advancing the release time of τ_i.

Therefore, the response time of τn is largest when it is released simultaneously with τi
… repeat the argument for all tasks…

RM is Optimal

- Based on the result from previous slide: the task set schedulability can be easily checked at the critical instants of the tasks in the set:
  - If all tasks are feasible at their critical instants, then the task set is schedulable in any other condition
- The optimality of RM is justified by showing that if a task set is schedulable by an arbitrary priority assignment, then it is also schedulable by RM.

Optimality by RM, 2 Tasks Set

- It can be shown that, given two periodic tasks $\tau 1$ and $\tau 2$, with $T1 < T2$, and if their schedule is feasible by an arbitrary priority assignment, then it is also feasible by RM.
  - That is, RM is optimal.
- This result can be further extended to a set of n periodic tasks.

2 Tasks Set Scheduled by a Different Algorithm

Consider a set of two periodic tasks τ1 and τ2, with T1 < T2.

This task set is schedulable by an algorithm different than RM

If priorities are not assigned according to RM, then task τ2 will receive the highest priority

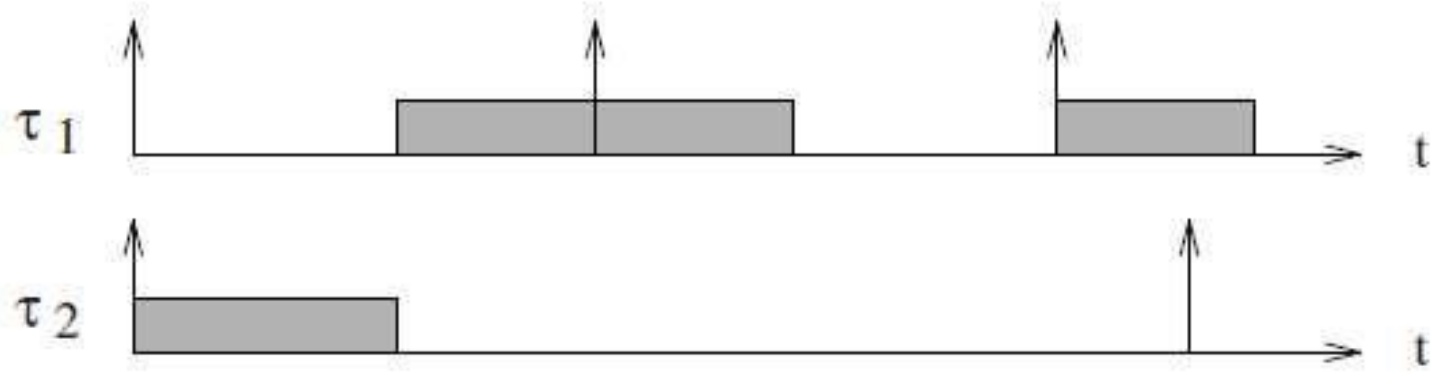


Figure 4.6 Tasks scheduled by an algorithm different from RM.

Here in fact we have a critical instant that we depicted before. As a result the schedule is feasible if the following inequality is satisfied:

$$C1 + C2 \leq T1.$$

2 Tasks Set Scheduled by RM

- If priorities are assigned according to RM, task τ1 will receive the highest priority.
- This situation is illustrated in next the slide
- In order to guarantee a feasible schedule two cases must be considered.
  - Let $F = \lfloor T_2 / T_1 \rfloor$ be the integer number of periods of T1 entirely contained in T2.
- Case (a): $C_1 < T_2 - F \cdot T_1$
- Case (b): $C_1 \geq T_2 - F \cdot T_1$

Proof: … in class work …

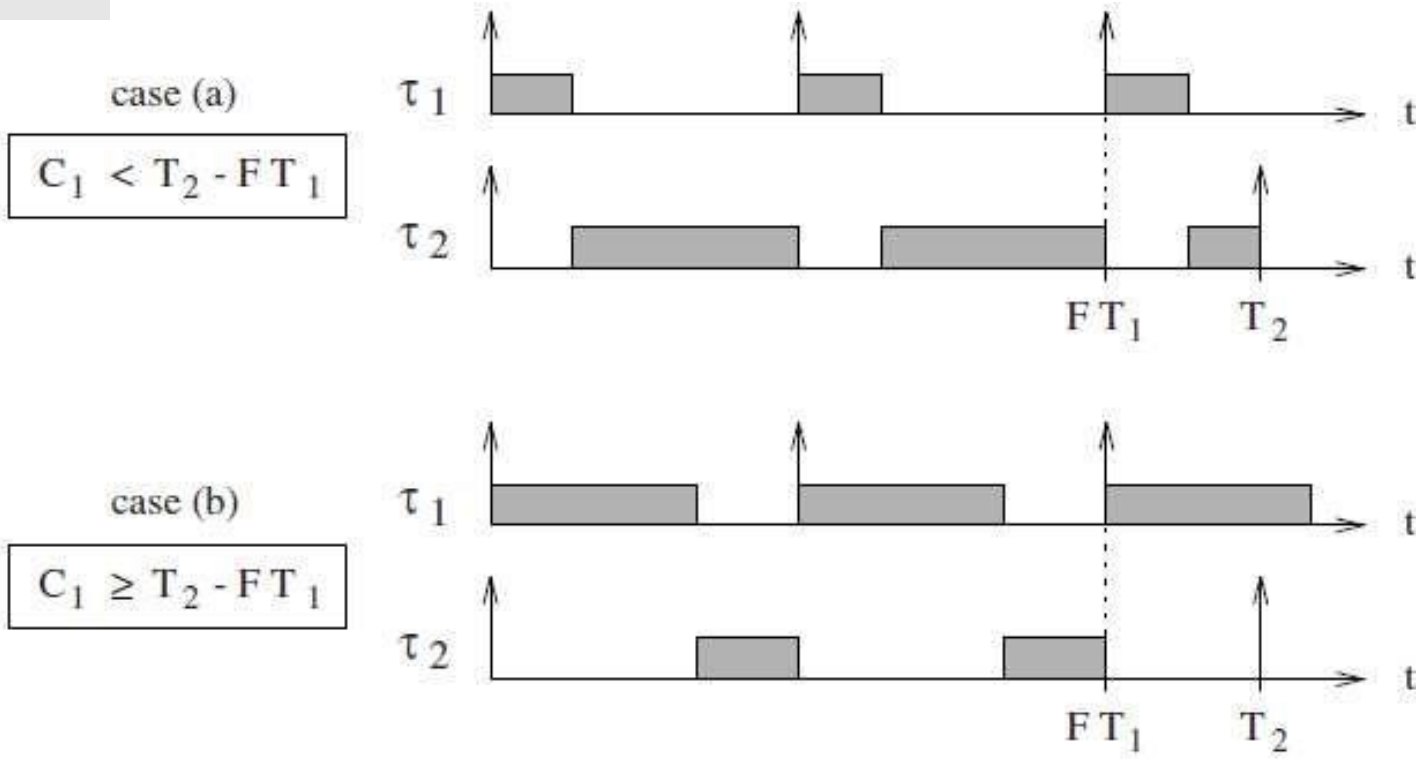


Figure 4.7 Schedule produced by RM in two different conditions.

RM Optimality

- We shown that, given two periodic tasks τ1 and τ2, with T1 < T2, 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.
- Next we show how to compute the least upper bound U_{lub} of the processor utilization factor for the RM algorithm.
  - This is an important result for task set schedulability analysis
  - The bound is first determined for two tasks and then it can be extended for an arbitrary number of tasks.

Calculation of U_{lub} for Two Tasks Set

- Consider the set of two periodic tasks τ1 and τ2, with T1 < T2
- In order to compute U_{lub} for RM we must:
  - Assign RM priorities to the tasks; τ1 will have the highest priority
  - Compute the upper bound U_{ub} for the task set by inflating computation times to fully utilize the processor
  - Minimize U_{ub} with respect to all other task parameters
- Let F = $\lfloor T_2/T_1 \rfloor$
- The computation time C2 is adjusted to fully utilize the processor
- Two cases (as function of F) will be considered for calculating U_{ub}

Case 1

C1 < T2 − F·T1

In class work …
See pp. 90 in the reference book!

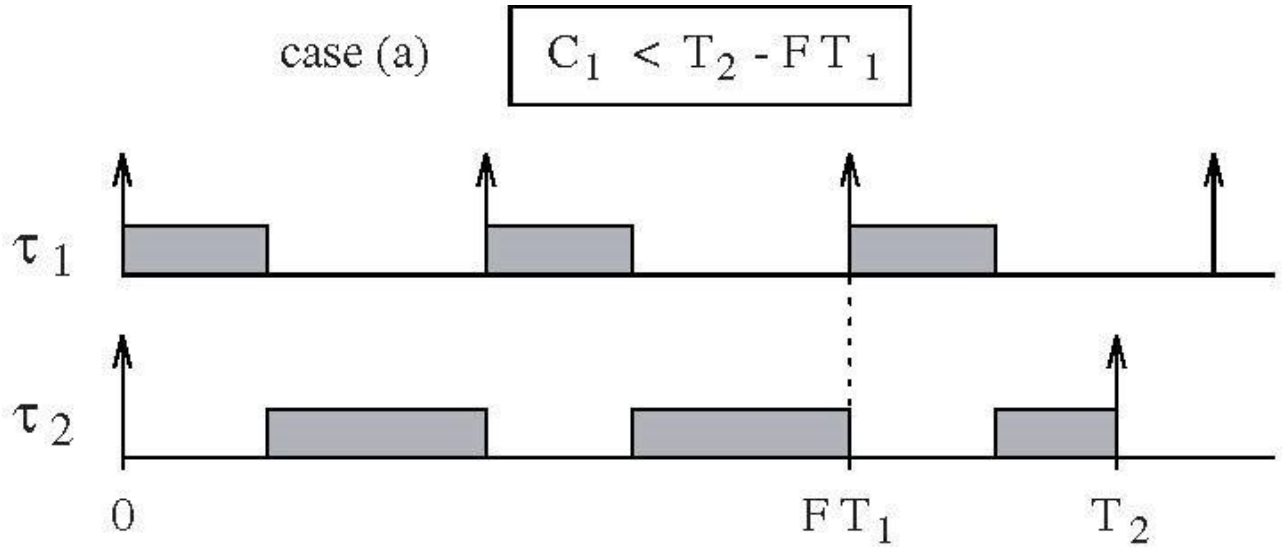


Figure 4.8 The second request of τ_2 is released when τ_1 is idle.

Case 2

$C1 \geq T2 - F \cdot T1$

In class work … See pp. 91 in the reference book!

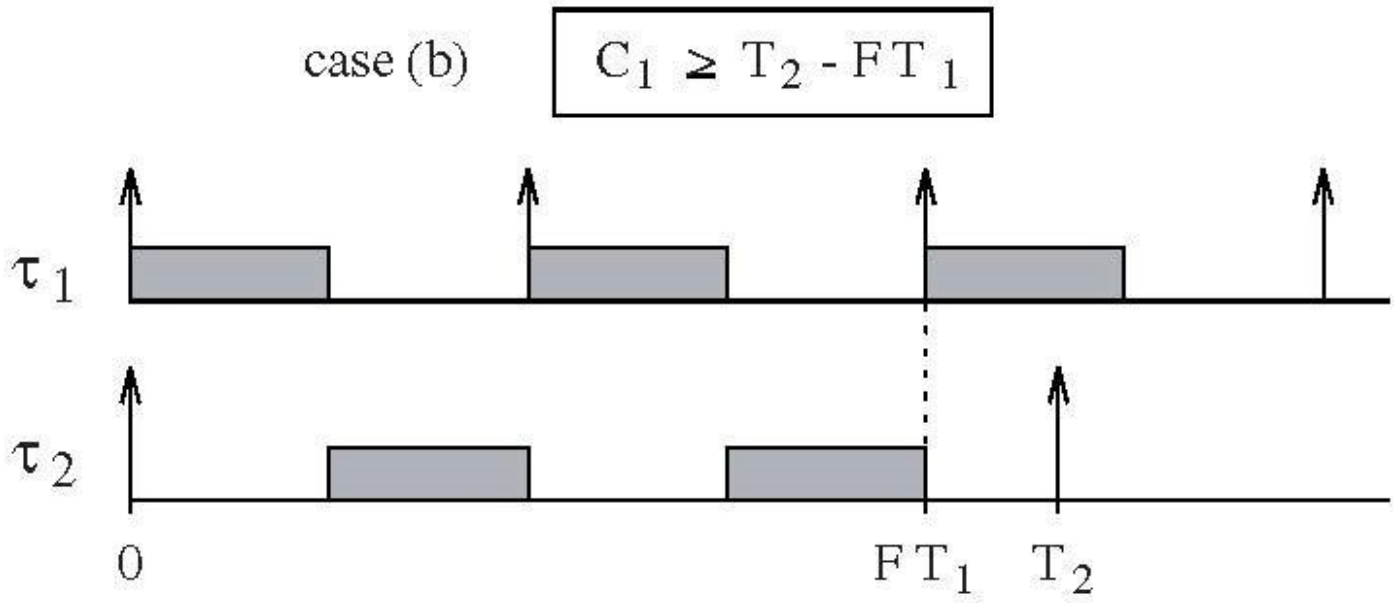


Figure 4.9 The second request of τ_2 is released when τ_1 is active.

Calculation of U_{LUB} for Two Tasks, RM

2 Tasks Set Minimum U_{ub} (See pp. 92 to 93)

- In both cases, the minimum value of U_{ub} occurs for:

  $C1 = T2 - T1 \times F$

- Expressing U from previous cases as U(F) we get the minimum of U occurring for the minimum value of F; namely, F = 1. Thus:

  $U = \frac{1+G^2}{1+G}$; where $G = T_2/T_1 - F$;

- Now minimizing U over G we get:

  $U_{lub} = 2(2^{0.5} - 1) \cong 0.83;$

Analysis for a Given F Value

- If T2 is a multiple of T1 then G = 0 => U = 1
- In general the utilization factor for two tasks can be computed as a function of the ratio k = T2/T1;
- For a given F, minimizing U for k we get
  - U* = …

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

Table 4.1 Values of k_i^* and U_i^* as a function of F.

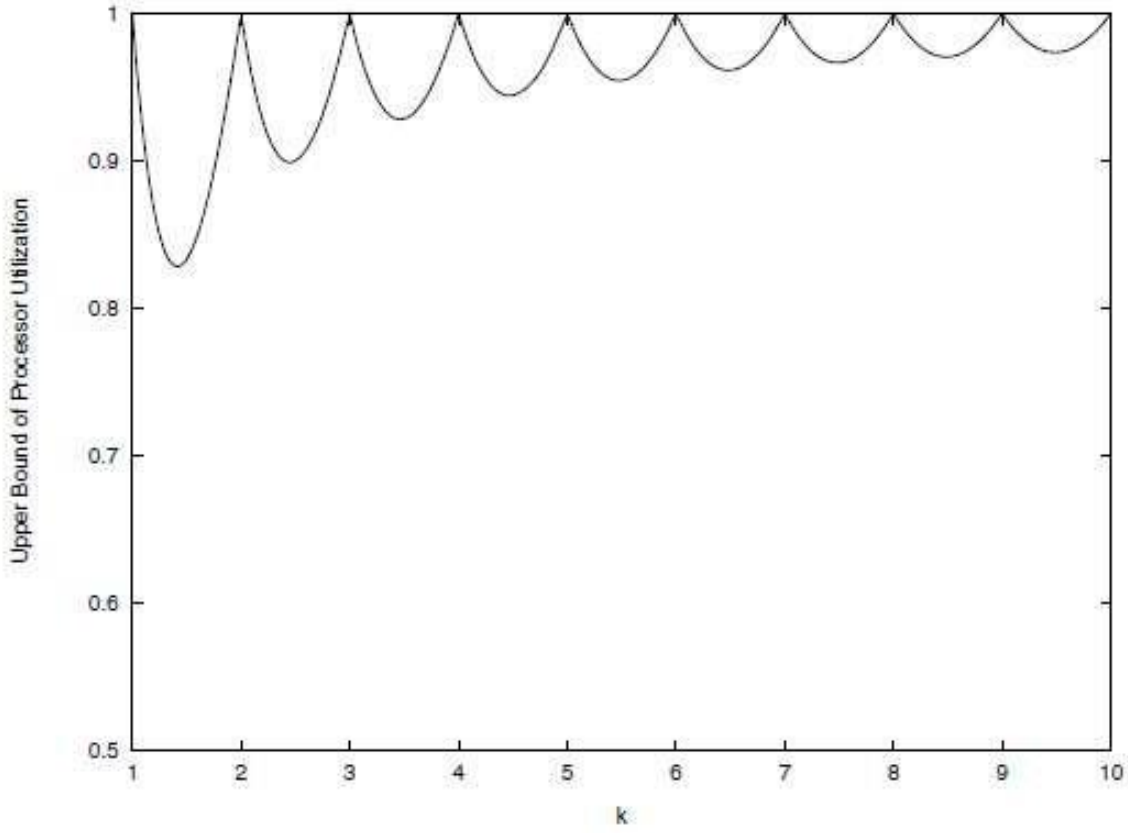


Figure 4.10 Upper bound of the processor utilization factor as a function of the ratio $k = T_2/T_1$.

RM U_{lub} Equation for n Tasks (See pp.95-96)

The results shown for a 2 tasks set can be generalized as follows:

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

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

$$\lim_{n \to \infty} U_{lub}(n) = \ln 2.$$

Hyperbolic Bound for RM

- The feasibility analysis of the RM algorithm can also be performed using the Hyperbolic Bound approach.
- The following theorem provides a sufficient condition for testing the schedulability of a task set under the RM algorithm.
- **Theorem 4.1** Let Γ = {τ1, . . . , τn} be a set of n periodic tasks, where each task τi is characterized by a processor utilization Ui. Then, Γ is schedulable with the RM algorithm if:

$$\prod_{i=1}^{n}(U_i + 1) \leq 2. \qquad (4.10)$$

$$\prod_{i=1}^{n}(U_i + 1) \leq 2. \qquad (4.10)$$

… proof, in class work …

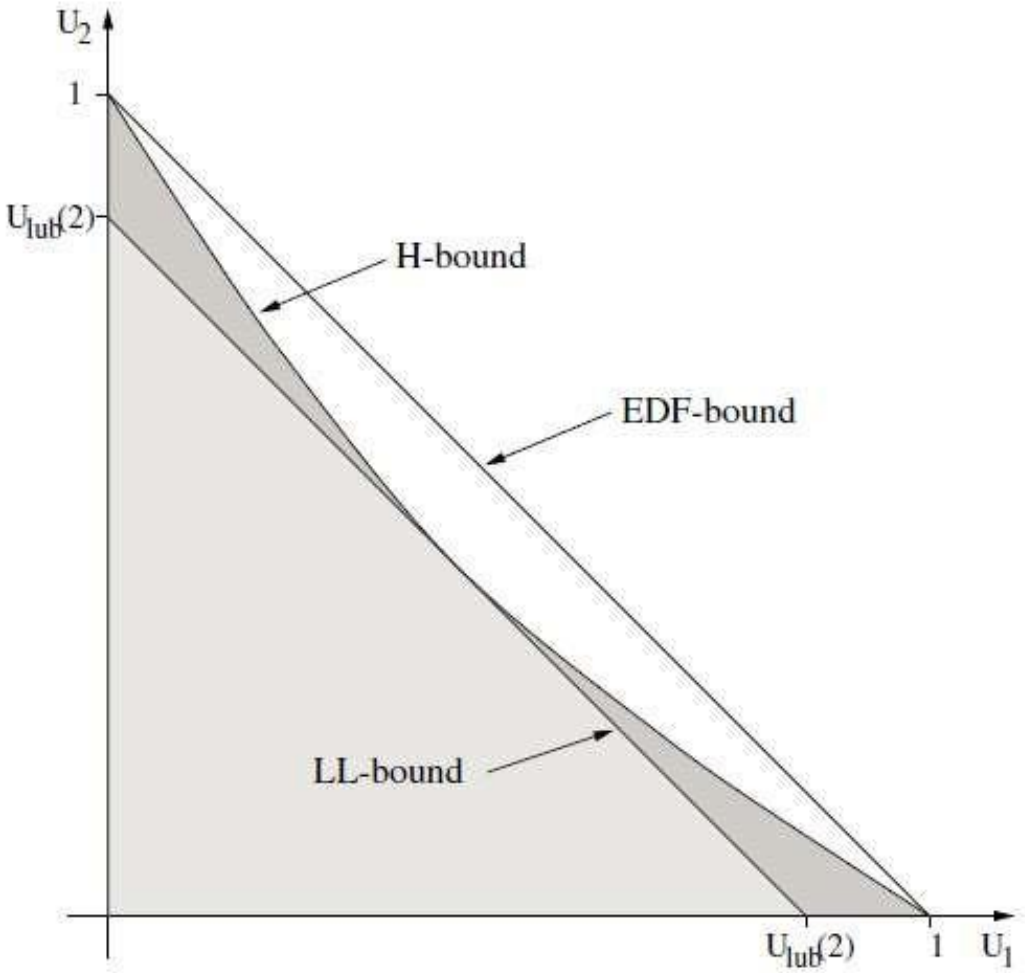


Figure 4.11 Schedulability bounds for RM and EDF in the utilization space.

Earliest Deadline First

- The Earliest Deadline First (EDF) algorithm is a dynamic scheduling rule using the absolute deadlines such that, 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: $d_{i,j} = \Phi_i + (j - 1)T_i + D_i$, EDF is a dynamic priority assignment.
- EDF, it is typically executed in preemptive mode.
- Also, EDF does not make any specific assumption on the periodicity of the tasks; works for both, aperiodic and periodic
  - The proof of optimality is the same as presented in Chapter 3.

Schedulability Analysis

- 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.

Theorem 4.2 *A set of periodic tasks is schedulable with EDF if and only if*

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

- See proof pp. 100 to 101 in the book

Notes on Proof of EDF Schedulability

Follows two steps:

1. Only if. We show that a task set cannot be scheduled if:

$$U > 1$$

2. If. We show the sufficiency by contradiction; assume that the condition $U < 1$ is satisfied and yet the task set is not schedulable.

Review proof in the textbook.

An Example

- The periodic task set of Figure 4.13:

U = 2/5 + 4/7 = 34/35 = 0.97;

- Then, 97 % of the processor time is used to execute the periodic tasks, while the CPU is idle in the remaining 3 %.
- Since U > 2(√2−1) = 0.83, the schedulability of the task set cannot be guaranteed under RM, whereas it is guaranteed under EDF.
- RM unfeasible schedule

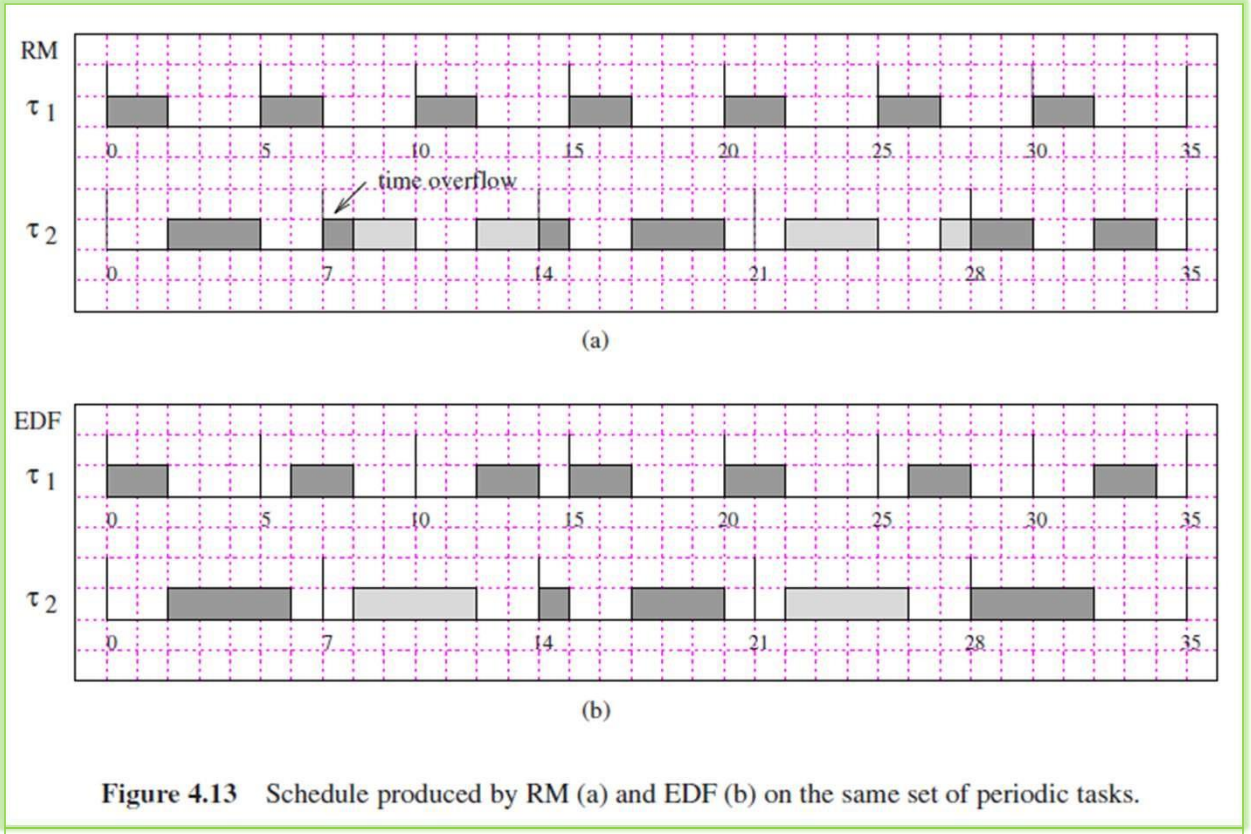


Figure 4.13 Schedule produced by RM (a) and EDF (b) on the same set of periodic tasks.

The smaller number of preemptions in EDF is a direct consequence of the dynamic priority assignment, which at any instant privileges the task with the earliest deadline, independently of tasks' periods.

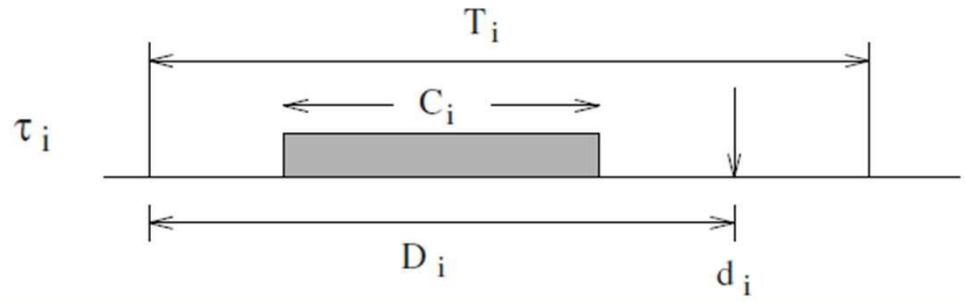
Deadline Monotonic (DM)

- DM is an extension of the RM where tasks can have relative deadlines less than or equal to their period
  - Assumption A3 is relaxed
- Each periodic task τi is characterized by:

- 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.

These parameters are illustrated in Figure 4.14 and have the following relationships:

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



DM Algorithm

- According to the DM algorithm, each task is assigned a fixed priority Pi inversely proportional to its relative deadline Di.
  - As a result, 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 on in a fully preemptive mode
- The DM priority assignment is optimal, meaning that, if a task set is schedulable by some fixed priority assignment then is also schedulable by DM
- The proof of DM optimality is like the one done for RM

DM Schedulability Test

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

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

- The test above is quite pessimistic, since the workload on the processor would be overestimated.

Less Pessimistic Schedulability Test for DM

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.

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:

$$\forall i : 1 \leq i \leq n \quad C_i + I_i \leq D_i, \tag{4.16}$$

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:

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

Cont …

- ✓ This test is sufficient but not necessary for guaranteeing the schedulability of the task set.
- ✓ The actual interference can be smaller than I_i, since τ_i may terminate earlier

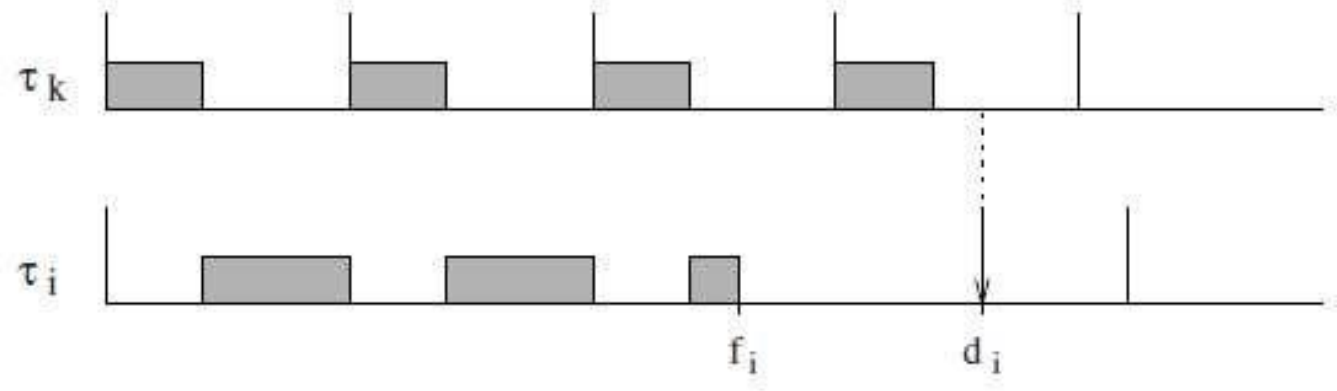


Figure 4.15 More accurate calculation of the interference on τ_i by higher priority tasks.

- To find a sufficient and necessary schedulability test for DM, the exact interleaving of higher-priority tasks must be evaluated for each process.
- But this analysis is quite costly.

Response Time Analysis

- However, the Response Time Analysis is an efficient method for evaluating the exact interference on periodic tasks and provides a sufficient and necessary schedulability test for DM

Response Time Analysis

- According to this method the longest response time Ri of a periodic task τi is computed, at the critical instant, as the sum of its computation time and the interference I_i of the higher priority tasks:

$$R_i = C_i + I_i,$$

where

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

Hence,

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

No simple solution exists for this equation since R_i appears on both sides.

To solve Ri equation we should note that, only a subset of points in [0, D_i] need to be checked for this test as given by this algorithm.

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 $[0, R_i^{(k)}]$:

$$I_i^{(k)} = \sum_{j=1}^{i-1} \left\lceil \frac{R_i^{(k)}}{T_j} \right\rceil C_j. \tag{4.18}$$

Then 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 equation (4.18).
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 $R_i \leq D_i$.

Example of Response Test Analysis

- Consider the set of periodic tasks shown in Table simultaneously activated at time t = 0.

- In order to guarantee τ4, we have to calculate R4 and verify that R4 ≤ D4.

	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

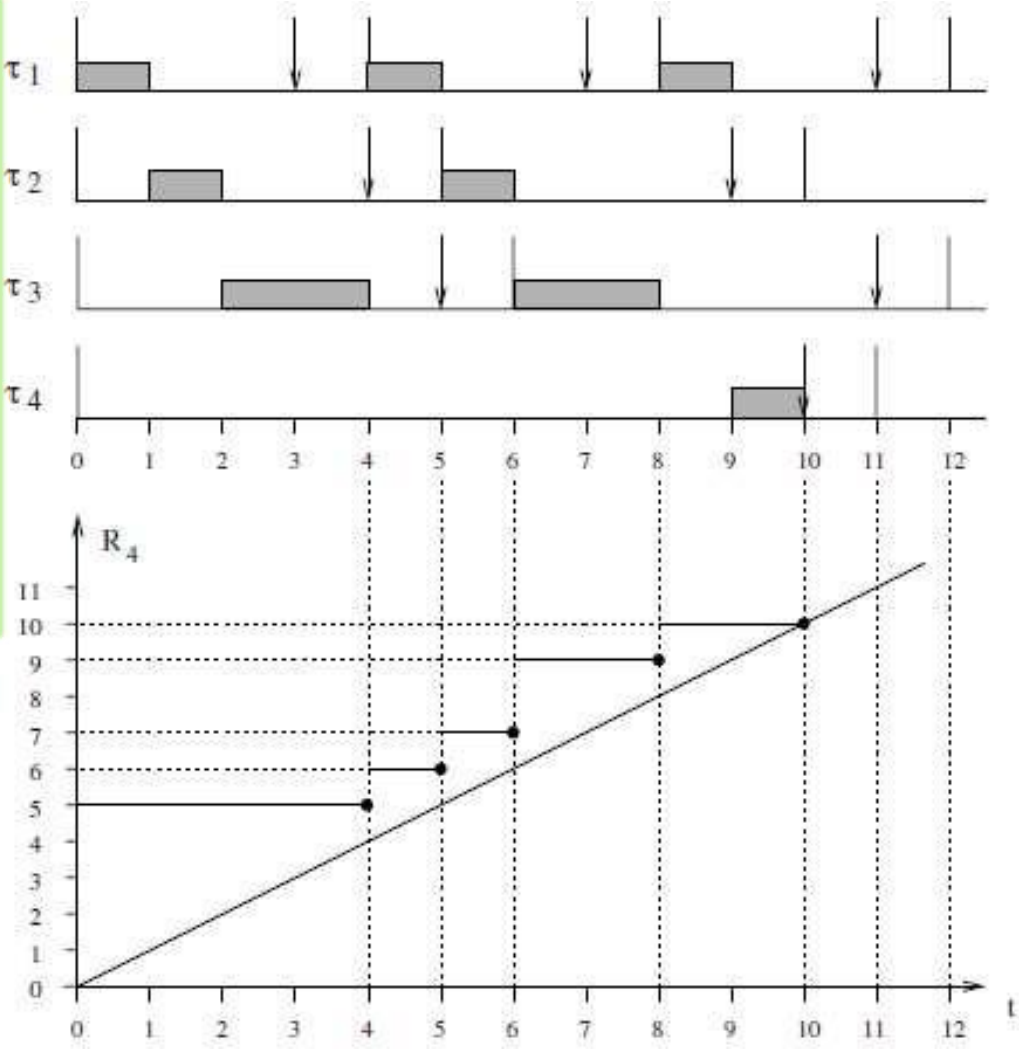
Step 0: $R_4^{(0)} = \sum_{i=1}^{4} C_i = 5$, but $I_4^{(0)} = 5$ and $I_4^{(0)} + C_4 > R_4^{(0)}$
hence τ_4 does not finish at $R_4^{(0)}$.

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 R4 ≤ D4, τ4 is schedulable within its deadline.

If Ri ≤ Di for all tasks, we conclude that the task set is schedulable by DM.

Schedulability Test Algorithm for DM

If Ri ≤ Di for all tasks, we conclude that the task set is schedulable by DM.

Such a schedulability test can be performed by the algorithm

```
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);
}
```

Another Schedulabitity Test

- Workload analysis test can also be used to provide a necessary and sufficient test for checking the schedulability of fixed priority systems with constrained deadlines (See Section 4.5.3 in the reference book)

EDF with Constrained Deadlines

- Under EDF (dynamic priority assignment system), the analysis of periodic tasks with deadlines less than or equal to periods can be performed using the processor demand criterion (See Section 4.6).

Comparison Between RM and EDF, Advantages

- We presented fixed (RM) and dynamic (EDF) priority assignments for the problem of scheduling a set of independent and preemptable periodic tasks.
- Advantage comparisons:
- Fixed priority approach -- simpler to implement.
  - If the ready queue is implemented as a multi-level queue with P priority levels (where P is the number of different priorities in the system) both task insertion and extraction can be achieved in O(1) complexity.
- Dynamic priority scheme – is in general better with respect to a fixed priority algorithm
  - Except for very large task sets (consisting of hundreds of tasks), or for very slow processors, where the ready queue implementation might give advantages in regard to reduce overheads.

Comparison Between RM and EDF, Schedulability

- Schedulability analysis, for the simple case of independent tasks with relative deadlines equal to periods
  - RM -- guarantee test requires a pseudo-polynomial complexity, even in the simple case of independent tasks with relative deadlines equal to periods,
  - EDF – guarantee test can be performed in O(n) for EDF.
- Schedulability analysis in the general case in which deadlines can be less than or equal to periods,
  - RM and EDF -- schedulability analysis becomes pseudo-polynomial for both algorithms.
  - Under fixed-priority assignments, the feasibility of the task set can be tested using the response time analysis,
  - Under dynamic priority assignments it can be tested using the processor demand criterion.

Comparison Between RM and EDF, Processor Utilization

- Processor utilization criteria
  - EDF is able to exploit the full processor bandwidth
  - RM algorithm can only guarantee feasibility for task sets with utilization less than 69%, in the worst case.
    - Other statistical results show that RM algorithm is able to feasibly schedule task sets with a processor utilization up to about 88%
- EDF will need extra computations for updating the absolute deadline at each job activation, however, EDF introduces less runtime overhead than RM, when context switches are taken into account.
- In fact, to enforce the fixed priority order, the number of preemptions that typically occur under RM is much higher than under EDF.

Homework

- Exercises 4.1 to 4.7