DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Task Based Scheduling

- Fixed and variable priorities
- Critical Instant
- Rate Monotonic (RM) scheduling
- Earliest Deadline First (EDF) scheduling

Fixed and variable priorities in Real Time scheduling

- Variable (dynamic) priority assigned to tasks is normally performed in non real time systems in order to make the system more “fluid”
    - Based on the concept that a task that is going to use the processor for the shortest time should take it first.
    - This maximizes the number of tasks executed per unit of time

- Fixed priority assignment favorises “important” tasks that must be executed first

- In Linux and Windows tasks above a given priority are not subject to dynamic priority management

- Dynamic priorities are also used in real-time systems
    - On the one side they are more complicated to implement
    - On the other side, the processor utilization is better exploited

Critical instant

- The “critical instant,” that is, the “**worst**” situation that may occur when a set of periodic tasks with given periods and computation times is scheduled.

- Task jobs can in fact be released at arbitrary instants within their period, and the time between period occurrence and job release is called the **phase** of the task.

- We shall see that the worst situation will occur when all the jobs are initially released at the same time (i.e., when the phase of all the tasks is zero).

- Proving that the system under consideration is schedulable in such a bad situation means proving that it will be schedulable for every task phase.

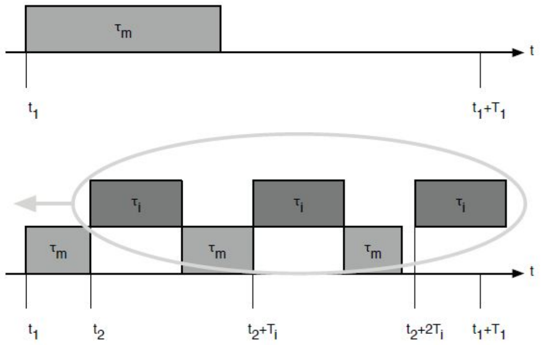
Critical instant theorem

- The relative deadline of a task is equal to its period, that is, $D_i = T_i \forall i$.
- Hence, for each task instance, the absolute deadline is the time of its next release, that is, $d_{i,j} = r_{i,j+1}$.
- We shall say that there is an overflow at time t if t is the deadline for a job that misses the deadline.
- A scheduling algorithm is ***feasible*** for a given set of task if they are scheduled so that no overflows ever occur.
- A ***critical instant*** for a task is an instant at which the release of the task will produce the largest response time.
- A ***critical time zone*** for a task is the interval between a critical instant and the end of the task response.

- **Theorem**: ***A critical instant for any task occurs whenever it is released simultaneously with the release of all higher-priority tasks.***

Critical instant theorem - proof

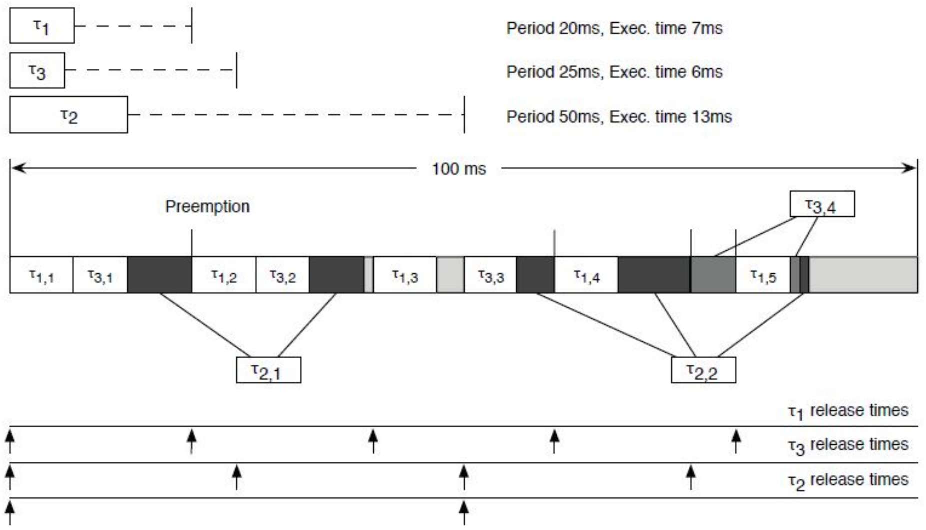
- If τ_m is released at t_1, between t_1 and $t_1 + T_m$, the time of the next release of τ_m, other tasks with a higher priority will possibly be released and interfere with the execution of τ_m because of preemption.



- It can be easily observed that if we anticipate the release of τ_m the interference due to higher priority tasks will never lower
- As a consequence the critical instant corresponds to phase zero, i.e. the task is release at the same time in respect of higher priority tasks

Rate Monotonic (RM) priority assignment

- Rate monotonic is a policy for fixed-priority assignment in periodic tasks, which assigns a priority that is inversely proportional to the task period: the shorter the task period, the higher its priority.
- Consider, for example, the three tasks: task τ1, which has the smallest period, will have the highest priority, followed by tasks τ3 and τ2, in that order.



Se funziona per un tempo uguale al LCM dei periodi poi si ripete uguale e funziona sempre

Optimality or RM

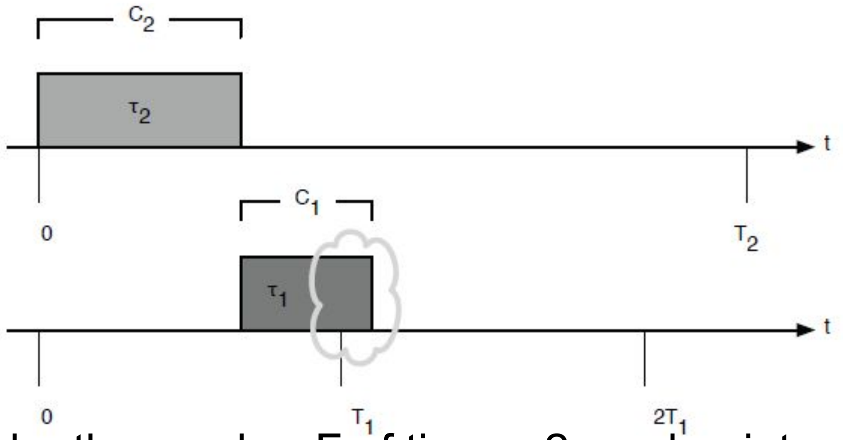
- Assumptions:
  - Every task τi is periodic with period Ti.
  - The relative deadline Di for every task τi is equal to its period Ti.
  - Tasks are scheduled preemptively and according to their priority.
  - There is only one processor

- RM is the optimal priority assignment policy, i.e. every system that is schedulable using fixed priorities is schedulable using RM policy

- In other words, if a system is not schedulable under RM policy, then it will remain not schedulable under any other fixed priority assignment

Proof - non RM priority assignment

- We consider two Tasks: τ1and τ2 with period T1 and T2 with T1 < T2
- In RM priority of τ1 is larger than that of τ2
- Consider the critical instant in which τ1 and τ2 are released together.
- When non RM priority of τ2 is larger and the schedulability condition is **C1 + C2 < T1**

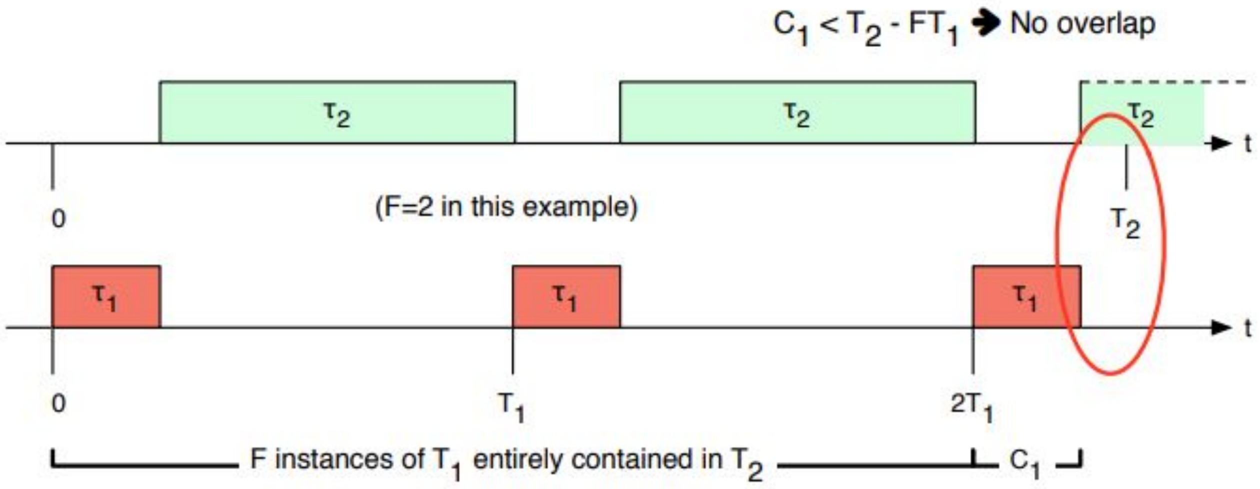


- Otherwise we consider the number F of times τ2 can be interrupted by τ1. The following lemma states the above schedulability condition implies the schedulability condition in RM

$$F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$$

Lemma- RM priority assignment

First Case



- The first case occurs when:

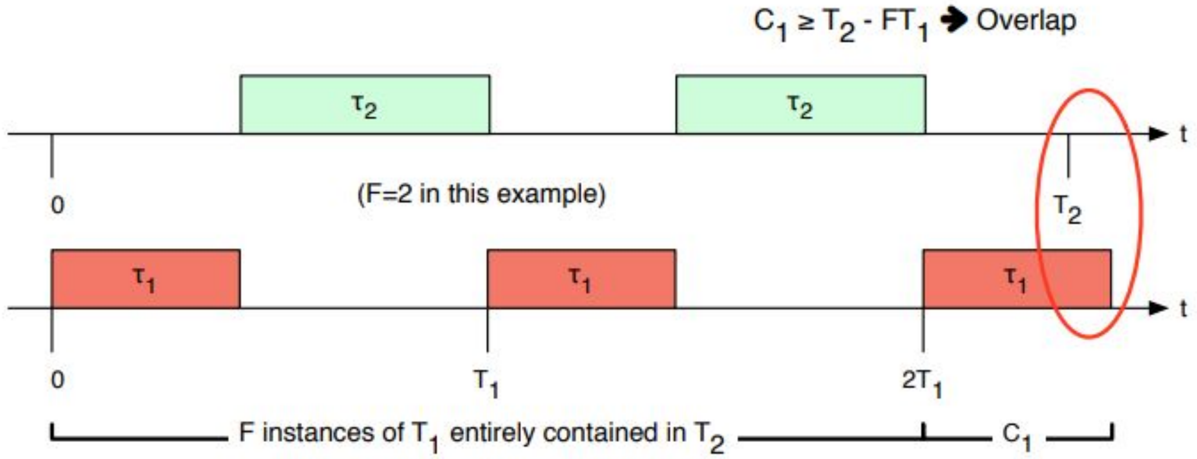
$$C_1 < T_2 - FT_1$$ uguale a dire F*T1+C1<T2

- From the figure, we can that the task set is schedulable if and only if:

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

Lemma- RM priority assignment

Second Case



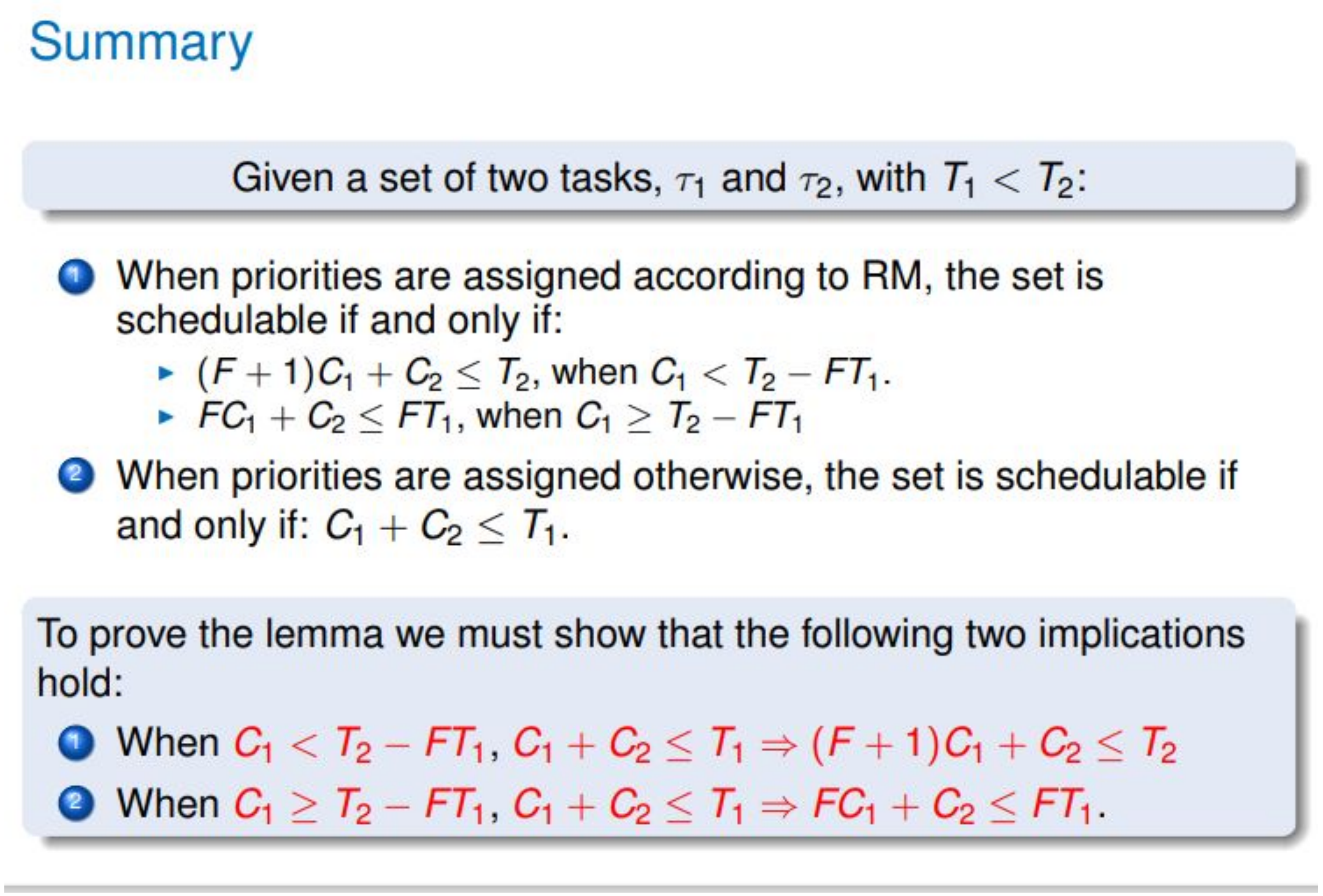
- The second case occurs when:

$$C_1 \geq T_2 - FT_1$$

- From the figure, we can see that the task set is schedulable if and only if:

$$FC_1 + C_2 \leq FT_1$$

Lemma - RM priority assignment



Summary

Given a set of two tasks, τ_1 and τ_2, with $T_1 < T_2$:

1. When priorities are assigned according to RM, the set is schedulable if and only if:
   - $(F+1)C_1 + C_2 \leq T_2$, when $C_1 < T_2 - FT_1$.
   - $FC_1 + C_2 \leq FT_1$, when $C_1 \geq T_2 - FT_1$
2. When priorities are assigned otherwise, the set is schedulable if and only if: $C_1 + C_2 \leq T_1$.

To prove the lemma we must show that the following two implications hold:

1. When $C_1 < T_2 - FT_1$, $C_1 + C_2 \leq T_1 \Rightarrow (F+1)C_1 + C_2 \leq T_2$
2. When $C_1 \geq T_2 - FT_1$, $C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FT_1$.

Lemma - RM priority assignment

Proving the First Implication

When $C_1 < T_2 - FT_1$, $C_1 + C_2 \leq T_1 \Rightarrow (F+1)C_1 + C_2 \leq T_2$

- If we multiply both members of $C_1 + C_2 \leq T_1$ by F and then add C_1, we obtain:

$$(F+1)C_1 + FC_2 \leq FT_1 + C_1$$

- We know that $F \geq 1$ (otherwise, it would not be $T_1 < T_2$), hence:

$$FC_2 \geq C_2$$

- Moreover, from the hypothesis we have:

$$FT_1 + C_1 < T_2$$

- As a consequence, we have:

$$(F+1)C_1 + C_2 \leq (F+1)C_1 + FC_2 \leq FT_1 + C_1 \leq T_2$$

Lemma - RM priority assignment

Proving the Second Implication

When $C_1 \geq T_2 - FT_1$, $C_1 + C_2 \leq T_1 \Rightarrow FC_1 + C_2 \leq FT_1$.

- It we multiply both members of of $C_1 + C_2 \leq T_1$ by F, we obtain:

$$FC_1 + FC_2 \leq FT_1$$

- We know that $F \geq 1$ (otherwise, it would not be $T_1 < T_2$), hence:

$$FC_2 \geq C_2$$

- As a consequence, we have:

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

- This concludes the proof of the lemma.

Extension to an arbitrary set of tasks

- **Theorem**: If the task set τ1, . . . , τn (n tasks) is schedulable by any arbitrary, but fixed, priority assignment, then it is schedulable by RM as well.

- The proof is a direct consequence of the previous considerations: let τi and τj be two tasks of adjacent priorities, τi being the higher-priority one, and suppose that Ti > Tj . Having adjacent priorities, both τi and τj are affected in the same way by the interferences coming from the higher-priority tasks (and not at all by the lower-priority ones). siccome hanno prorita' piu alta preemptano sempre quelle con piu bassa, da cui non vengono influenzate

- Hence, we can apply the result just obtained and state that if we interchange the priorities of τi and τj, the set is still schedulable.

- Finally, we notice that the RM priority assignment can be obtained from any other priority assignment by a sequence of pairwise priority reorderings as above, thus ending the proof.

Earliest Deadline First (EDF)

- The Earliest Deadline First (abbreviated as EDF) algorithm selects tasks according to their absolute deadlines. That is, at each instant, tasks with earlier deadlines will receive higher priorities.
    - Recall that the absolute deadline $d_{i,j}$ of the j-th instance (job) of the task τ_i is formally $d_{i,j} = \varphi_i + jT_i + D_i$ (12.14) where φ_i is the phase of task τ_i, that is, the release time of its first instance (for which $j = 0$), and T_i and D_i are the period and relative deadlines of task τ_i, respectively.
- The priority of each task is assigned dynamically, because it depends on the current deadlines of the active task instances.
- Task priorities may be updated only when a new task instance is released (task instances are released at every task period). Afterwards, when time passes, the relative order due to the proximity in time of the next deadline remains unchanged among active tasks, and therefore, priorities are not changed.
- As for RM, EDF is an intuitive choice as it makes sense to increase the priority of more "urgent" tasks, that is, for which deadline is approaching.

Optimality of EDF

- It can be proved that EDF is the optimal scheduling algorithm, that is, if any task set is schedulable by **any** scheduling algorithm, then it is also schedulable by EDF.

- This fact can be proved under the following assumption:
  - Tasks are scheduled preemptively;
  - There is only one processor.

- In practice EDF is more difficult ro implement and in this case the scheduler requires additional information about task deadlines that is normally not known