DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Schedulabilty Analysis based on Utilization and Response Time Analysis

- Computer utilization definition
- Sufficient Schedulability Test for Rate Monotonic (RM)
- Sufficient Schedulability Test for Earliest Deadline First (EDF)
- Response Time Analysis

Schedulability analysis

- We have analyzed two priority assignment policies: fixed priority and variable priority.
  - For fixed-priority scheduling, it has been shown that Rate Monotonic (RM) Scheduling is optimal
  - For variable-priority assignment, the optimality of Earliest Deadline First (EDF)
- Despite the elegance and importance of these two results, their practical impact for the moment is rather limited. In fact, what we are interested in practice is to know whether a given task assignment is schedulable, before knowing what scheduling algorithm to use.
- A sufficient condition for schedulability will be presented, which, when satisfied, ensures that the given set of tasks is definitely schedulable.
- The schedulability check will be very simple, being based on an upper limit in the processor utilization.
- This simplicity is, however, paid for by the fact that this condition is only a sufficient one.
  - As a consequence, if the utilization check fails, we cannot state that the given set of tasks is not schedulable.

Processor Utilization definition

- In the following,it is assumed that the basic process model is being used and, in particular, we shall consider single-processor systems.
- Given a set of N periodic tasks $\Gamma = \{\tau_1, \ldots, \tau_N\}$, the processor utilization factor **U** is the fraction of processor time spent in the execution of the task set, that is

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

- where C_i/T_i is the fraction of processor time spent executing task τ_i.

- The processor utilization factor is therefore a measure of the computational load imposed on the processor by a given task set and can be increased by increasing the execution times C_i of the tasks.

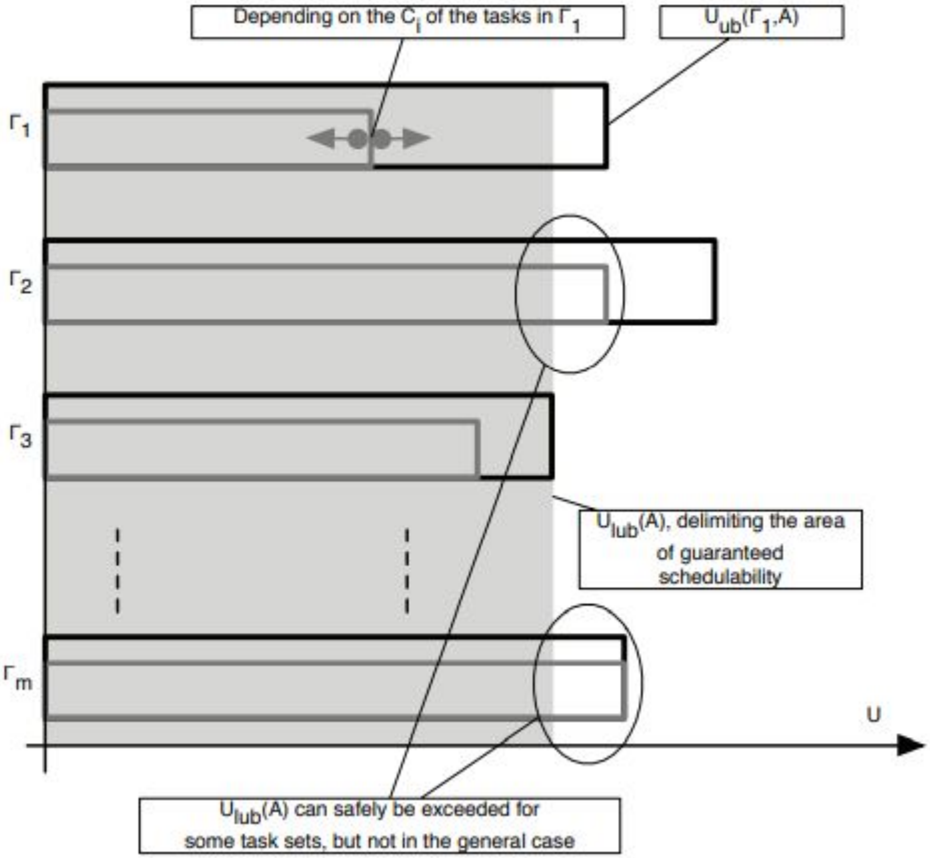
Least UpperBound (U_{LUB})

- A task set Γ is said to fully utilize the processor with a given scheduling algorithm A if it is schedulable by A, but any increase in the computational load C_i of any of its tasks will make it no longer schedulable. The corresponding upper bound of the utilization factor is denoted as $\mathbf{U_{ub}(\Gamma, A)}$.

- If we consider now all the possible task sets Γ, it is interesting to ask how large the utilization factor can be in order to guarantee the schedulability of any task set Γ by a given scheduling algorithm A.

- In order to do this, we must determine the minimum value of $U_{ub}(\Gamma, A)$ over all task sets Γ that fully utilize the processor with the scheduling algorithm A. This new value, called least upper bound and denoted as $\mathbf{U_{lub}(A)}$, will only depend on the scheduling algorithm A and is defined as $U_{lub}(A) = \min_\Gamma\{U_{ub}(\Gamma, A)\}$ where Γ represents the set of all task sets that fully utilize the processor.

A pictorial representation of U_{lub}



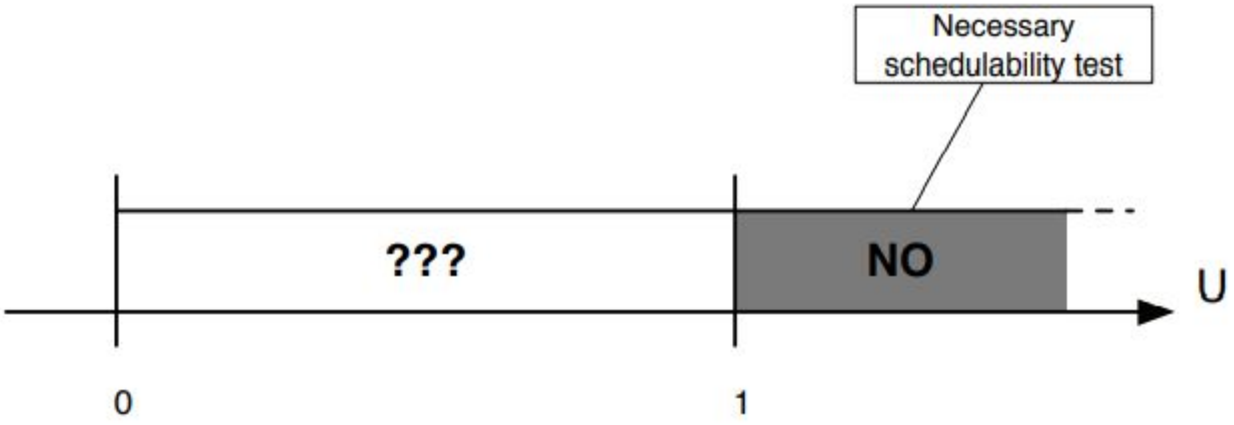
Schedulability

- For every possible task set Γ_i, the maximum utilization depends on both A and Γ_i.
- The actual utilization for task set Γ_i will depend on the computational load of the tasks but will never exceed $U_{ub}(\Gamma_i, A)$.
- Since $U_{lub}(A)$ is the minimum upper bound over all possible task sets, ***any task set whose utilization factor is below $U_{lub}(A)$ will be schedulable by A***.
- Observe that for a given task set Γ with scheduling algorithm A its utilization may exceed $U_{lub}(A)$ and be schedulable nevertheless, but this does not hold in general.
- On the other side, if the utilization factor U for a given task set Γ with scheduling algorithm A does not exceed $U_{lub}(A)$ we can for sure state that Γ is schedulable, i.e. no task will ever miss its deadline.
- This represents therefore a **sufficient** condition for schedulability

An upper limit of U

- If the processor utilization factor U of a task set Γ is greater than one (that is, if U > 1), then the task set is not schedulable, regardless of the scheduling algorithm.

- This result is intuitive, a set of tasks cannot require more than 100% cpu time in order to be executes, regardless the chosen scheduling algorithm

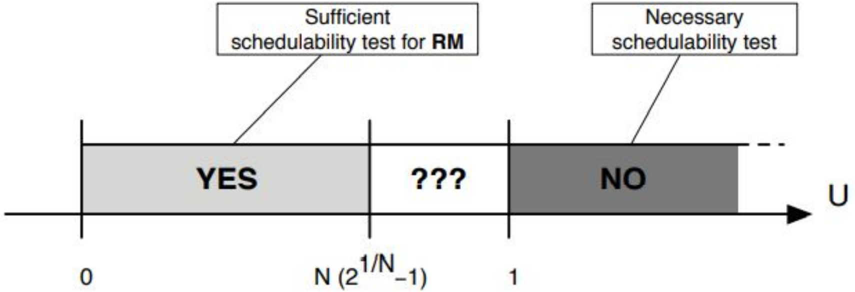


A sufficient condition for Rate Monotonic

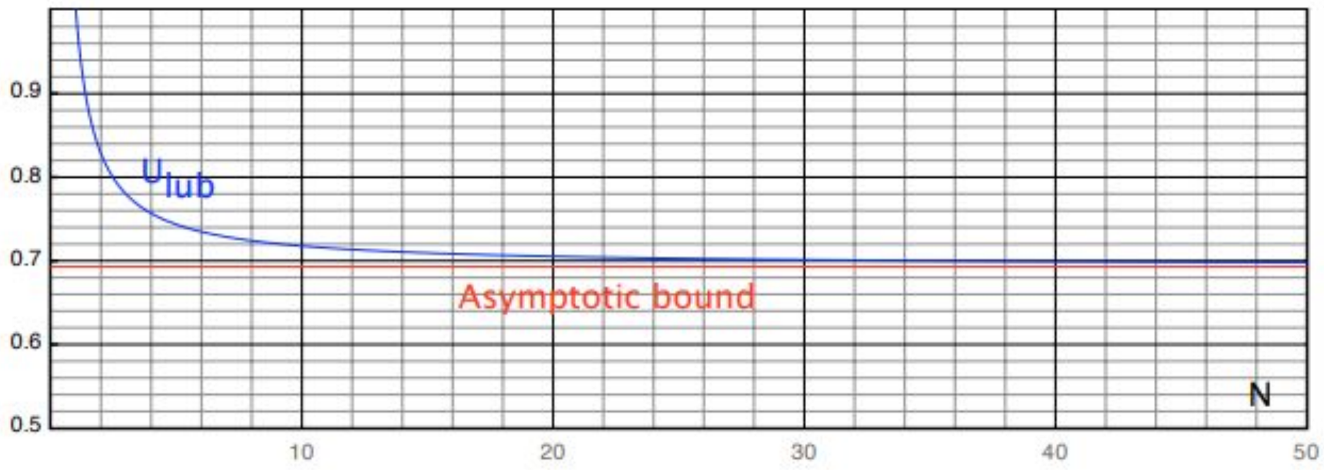
- **Theorem**: For a set of N periodic tasks scheduled by the Rate Monotonic algorithm, the least upper bound of the processor utilization factor Ulub is

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

- Recalling that a sufficient schedulability condition for a given set of tasks with Processor Utilization U and scheduling algorithm A is that U is not greater than $U_{lub}(A)$, this result provides a sufficient schedulability condition for RM



One step further



- U_{lub} is monotonically decreasing with respect to N.
- For large values of N, it asymptotically approaches ln 2 ≈ 0.693.
- From this observation a simpler — but more pessimistic — sufficient test can be stated: regardless of N, any task set with a combined utilization factor of less than ln 2 will always be schedulable by the Rate Monotonic algorithm.

Examples (1)

A task set definitely schedulable by RM.

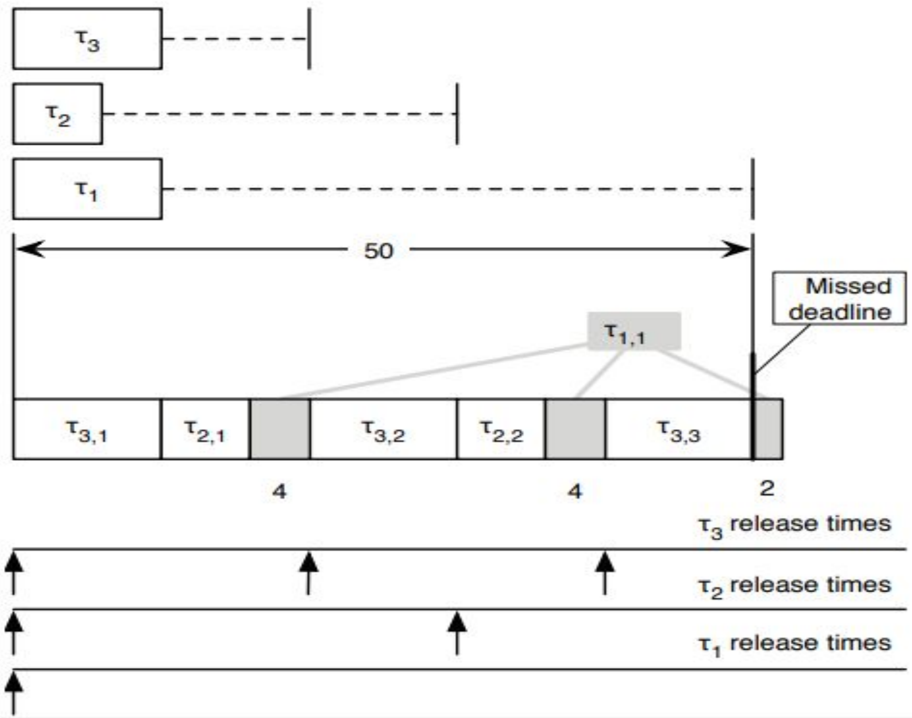
Task τ_i	Period T_i	Computation Time C_i	Priority	Utilization
τ_1	50	20	Low	0.400
τ_2	40	4	Medium	0.100
τ_3	16	2	High	0.125

- The processor Utilization for this task Set is 0.4+0.1+0.125 = 0.625 < ln2 ≈ 0.693
- We can therefore definitely state that this set of tasks is schedulable under RM scheduling

Examples (2)

A task set for which the sufficient RM scheduling condition does not hold.

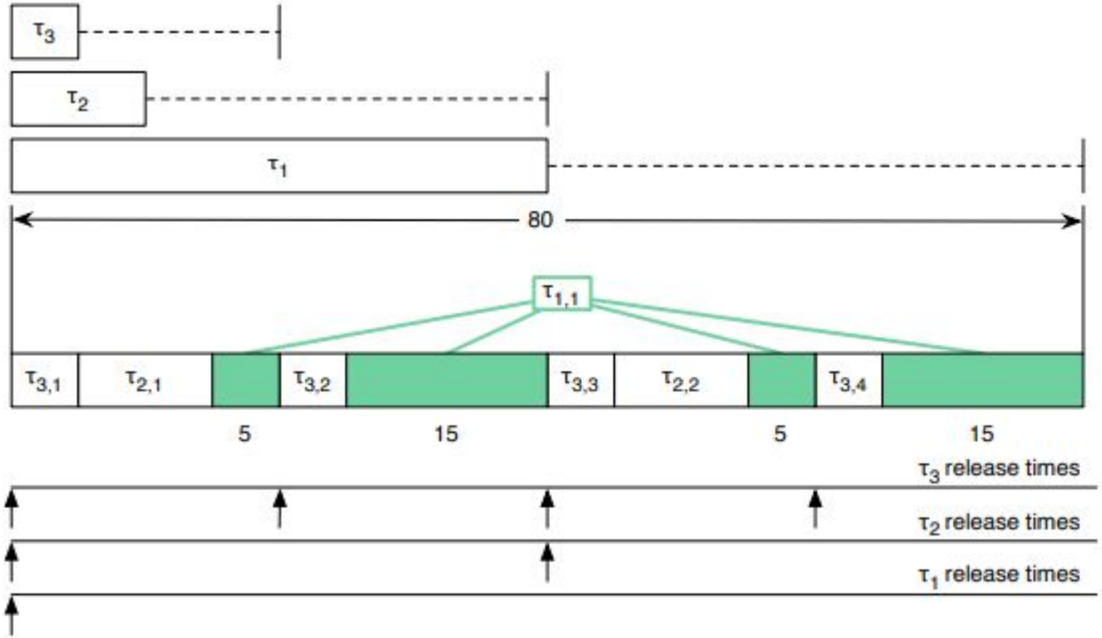
Task τ_i	Period T_i	Computation Time C_i	Priority	Utilization
τ_1	50	10	Low	0.200
τ_2	30	6	Medium	0.200
τ_3	20	10	High	0.500



Examples(3)

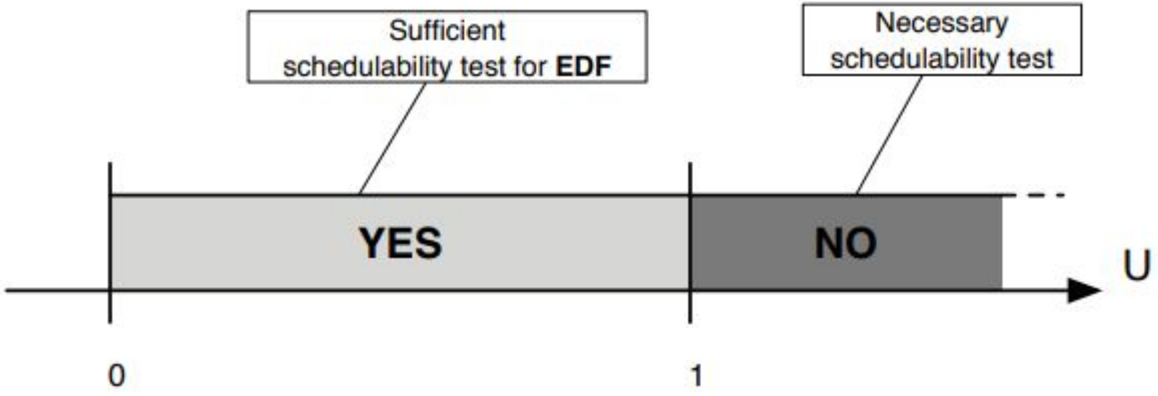
schedulabile anche
se non e' sotto il least upper bound

Task τ_i	Period T_i	Computation time C_i	Priority	Utilization
τ_1	80	40	Low	0.500
τ_2	40	10	Medium	0.250
τ_3	20	5	High	0.250



Schedulability condition for EDF

- Theorem: A set of N periodic tasks is schedulable with the Earliest Deadline First algorithm if and only its Processor Utilization is not greater than 1



Response Time Analysis (1)

- In this analysis the condition $D_i = T_i$ assumed before is now relaxed into condition $D_i \le T_i$.
- During execution, the preemption mechanism grabs the processor from a task whenever a higher-priority task is released. For this reason, all tasks (except the highest-priority one) suffer a certain amount of interference from higher-priority tasks during their execution.
- Therefore, the worst-case response time R_i of task τ_i is computed as the sum of its computation time C_i and the worst-case interference Ii it experiences, that is, $Ri = C_i + I_i$
- Observe that the interference must be considered over any possible interval [t, t + Ri], that is, for any t, to determine the worst case.
- We already know, however, that the worst case occurs when all the higher-priority tasks are released at the same time as task τi. In this case, t becomes a critical instant and, without loss of generality, it can be assumed that all tasks are released simultaneously at the critical instant t = 0.

Response Time Analysis (2)

- The contribution of each higher-priority task to the overall worst-case interference will now be analyzed individually by considering the interference due to any single task τ_j of higher priority than τ_i.

- Within the interval $[0, R_i]$, τj will be released one (at t = 0) or more times. The exact number of releases can be computed by means of a ceiling function, as

$$\left\lceil \frac{R_i}{T_j} \right\rceil$$

- Since each release of τj will impose on τi an interference of Cj , the worst-case interference imposed on τi by τj is

$$\left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- This because if task τ_j is released at any time $t<R_i$, than its execution must have finished before R_i, as τ_j has a larger priority, and therefore, that instance of τ_j must have terminated before τ_i can resume.

Response Time Analysis (3)

- Let hp(i) denote the set of task indexes with a priority higher than τ_i . These are the tasks from which τ_i will suffer interference. Hence, the total interference endured by τ_i is

$$I_i = \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

- Recalling that $R_i = C_i + I_i$, we get the following recursive relation for the worst-case response time R_i of τ_i:

$$R_i = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

Response Time Analysis (3)

- No simple solution exists for this equation since Ri appears on both sides
- The equation may have more than one solution: the smallest solution is the actual worst-case response time. The simplest way of solving the equation is to form a recurrence relationship of the form

$$w_i^{(k+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{w_i^{(k)}}{T_j} \right\rceil C_j$$

- where w(k) i is the k-th estimate of Ri and the (k+ 1)-th estimate from the k-th in the above relationship. The initial approximation w(0) i is chosen by letting $w(0)_i = C_i$ (the smallest possible value of R_i). The succession $w(0)_i$, $w(1)_i$, ..., $w(k)_i$, ... is monotonically nondecreasing.

- Two cases are possible for the succession $w(0)_i$, $w(1)_i$, ..., $w(k)_i$, ...:
    - If the equation has no solutions, the succession does not converge, and it will be $w(k)_i > D_i$ for some k. In this case, τ_i clearly does not meet its deadline.
    - Otherwise, the succession converges to R_i, and it will be $w(k)_i = w(k-1)_i = R_i$ for some k. In this case, τ_i meets its deadline if and only if $R_i \le D_i$.

se la response time supera la deadline possiamo fermarci e dire che non e’ schedulabile

Example (1)

Task τ_i	Period T_i	Computation Time C_i	Priority
τ_1	8	3	High
τ_2	14	4	Medium
τ_3	22	5	Low

- The priority assignment is Rate Monotonic and the CPU utilization factor U is

$$U = \sum_{i=1}^{3} \frac{C_i}{T_i} = \frac{3}{8} + \frac{4}{14} + \frac{5}{22} \simeq 0.89$$

- The highest-priority task τ1 does not endure interference from any other task. Hence, it will have a response time equal to its computation time, that is, $R_1 = C_1$. In fact hp(1) = ∅ and, given $w(0)_1 = C_1$, we trivially have $w(1)_1 = C_1$. In this case, $C_1 = 3$, hence $R_1 = 3$ as well. Since $R_1 = 3$ and $D_1 = 8$, then $R_1 \le D_1$ and τ_1 meets its deadline.
- For τ_2, hp(2) = {1} and $w(0)_2 = C_2 = 4$. The next approximations of R_2 are

$$\begin{aligned} w_2^{(1)} &= 4 + \left\lceil \frac{4}{8} \right\rceil 3 = 7 \\ w_2^{(2)} &= 4 + \left\lceil \frac{7}{8} \right\rceil 3 = 7 \end{aligned}$$

Example (2)

- Since $w(2)_2 = w(1)_2 = 7$, then the succession converges, and $R_2 = 7$. In other words, widening the time window from 4 to 7 time units did not introduce any additional interference. Task τ_2 meets its deadline, too, because $R_2 = 7$, $D_2 = 14$, and thus $R_2 \leq D2$. For τ_3, hp(3) = {1, 2}. It gives rise to the following calculations:

$$
\begin{aligned}
w_3^{(0)} &= 5 \\
w_3^{(1)} &= 5 + \left\lceil \frac{5}{8} \right\rceil 3 + \left\lceil \frac{5}{14} \right\rceil 4 = 12 \\
w_3^{(2)} &= 5 + \left\lceil \frac{12}{8} \right\rceil 3 + \left\lceil \frac{12}{14} \right\rceil 4 = 15 \\
w_3^{(3)} &= 5 + \left\lceil \frac{15}{8} \right\rceil 3 + \left\lceil \frac{15}{14} \right\rceil 4 = 19 \\
w_3^{(4)} &= 5 + \left\lceil \frac{19}{8} \right\rceil 3 + \left\lceil \frac{19}{14} \right\rceil 4 = 22 \\
w_3^{(5)} &= 5 + \left\lceil \frac{22}{8} \right\rceil 3 + \left\lceil \frac{22}{14} \right\rceil 4 = 22
\end{aligned}
$$

- $R_3 = 22$ and $D_3 = 22$, and thus $R_3 \leq D_3$ and τ_3 (just) meets its deadline.

- In this case RTA guarantees that all tasks meet their deadline