Scheduling Periodic Real-Time Tasks on Uniprocessor Systems

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

07/13, Dec., 2016

Periodic Control System

Pseudo-code for this system

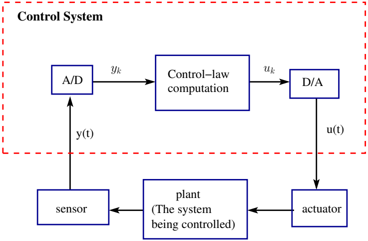
set timer to interrupt periodically with period T;

at each timer interrupt

do

- perform analog-to-digital conversion to get y;
- compute control output u;
- output u and do digital-to-analog conversion;

od



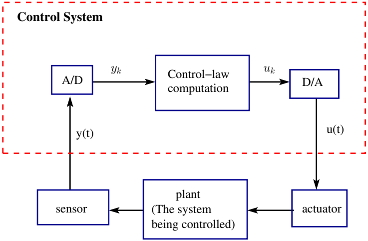
Example: Sporadic Control System

Pseudo-code for this system

while (true)

- start := get the system tick;
- perform analog-to-digital conversion to get y;
- compute control output u;
- output u and do digital-to-analog conversion;
- end := get the system tick;
- $timeToSleep :=$ $T - (end - start)$;
- sleep $timeToSleep$;

end while



Periodic and Sporadic Task Models

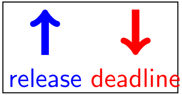
- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume ϕ_i is 0.

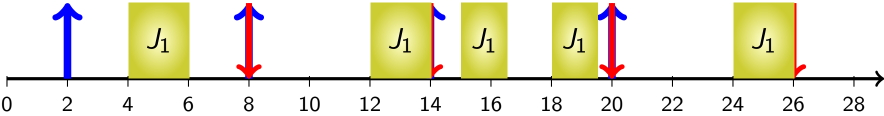
Periodic and Sporadic Task Models

- When jobs (usually with the same computation requirement) are released recurrently, these jobs can be modeled by a recurrent task
- Periodic Task τ_i:
  - A job is released exactly and periodically by a period T_i
  - A phase ϕ_i indicates when the first job is released
  - A relative deadline D_i for each job from task τ_i
  - (ϕ_i, C_i, T_i, D_i) is the specification of periodic task τ_i, where C_i is the worst-case execution time. When ϕ_i is omitted, we assume ϕ_i is 0.
- Sporadic Task τ_i:
  - T_i is the minimal time between any two consecutive job releases
  - A relative deadline D_i for each job from task τ_i
  - (C_i, T_i, D_i) is the specification of sporadic task τ_i, where C_i is the worst-case execution time.

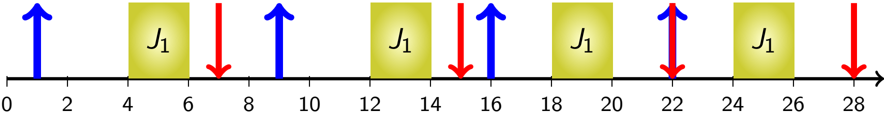
Examples of Recurrent Task Models

Periodic task: $(\phi_i, C_i, T_i, D_i) = (2, 2, 6, 6)$





Sporadic task: $(C_i, T_i, D_i) = (2, 6, 6)$



Relative Deadline <=> Period

For a task set, we say that the task set is with

- *implicit deadline* when the relative deadline D_i is equal to the period T_i, i.e., $D_i = T_i$, for every task τ_i,
- *constrained deadline* when the relative deadline D_i is no more than the period T_i, i.e., $D_i \leq T_i$, for every task τ_i, or
- *arbitrary deadline* when the relative deadline D_i could be larger than the period T_i for some task τ_i.

The response time of a job is its finishing time minus its arrival time. The worst-case response time of task τ_i is *the maximum response time* among the jobs of task τ_i.

Some Definitions for Sporadic/Periodic Tasks

- Periodic Tasks:
  - Synchronous system: Each task has a phase of 0.
  - Asynchronous system: Phases are arbitrary.
- Hyperperiod: Least common multiple (LCM) of T_i.
- Task utilization of task τ_i: $U_i := \frac{C_i}{T_i}$.
- System (total) utilization: $U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} U_i$.

Outline

Schedulability for Dynamic-Priority Scheduling

Schedulability for Static-Priority (or Fixed-Priority) Scheduling

Utilization-Based Test for EDF Scheduling

Theorem

Liu and Layland: A task set $\mathcal{T}$ of n independent, preemptable, periodic tasks with implicit deadlines can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most 100%.

Proof

- The *only if* part is obvious: If $U > 1$, then some task clearly must miss a deadline. So, we concentrate on the *if* part.
- Contrapositive: if $\mathcal{T}$ is not schedulable, then $U > 1$.
  - Let $J_{i,k}$ be the first job to miss its deadline
  - Let $d_{i,k}$ be the absolute deadline of $J_{i,k}$
  - Let t_{-1} be the last instant before $d_{i,k}$, at which the processor is either idle or executing a job with absolute deadline larger than $d_{i,k}$

(cont.)

Proof of Utilization-Bound Test for EDF

Proof.

Because $J_{i,k}$ missed its deadline, we know that

$$d_{i,k} - t_{-1} < \begin{array}{l}\text{demand in } [t_{-1}, d_{i,k}) \\ \text{by jobs with absolute deadline no more than } d_{i,k}\end{array}$$

$$= \sum_{j=1}^{n} \left\lfloor \frac{d_{i,k} - t_{-1}}{T_j} \right\rfloor C_j$$

$$\leq \sum_{j=1}^{n} \frac{d_{i,k} - t_{-1}}{T_j} C_j$$

By cancelling $d_{i,k} - t_{-1}$, we conclude the proof by

$$1 < \sum_{j=1}^{n} \frac{C_j}{T_j} = U.$$

Relative Deadlines Less than Periods

Theorem

A task set $\mathcal{T}$ of n independent, preemptable, periodic tasks with constrained deadlines can be feasibly scheduled (under EDF) on one processor if

$$\sum_{k=1}^{n} \frac{C_k}{\min\{D_k, T_k\}} \leq 1.$$

This theorem only gives a sufficient schedulability test.

Outline

Schedulability for Dynamic-Priority Scheduling

Schedulability for Static-Priority (or Fixed-Priority) Scheduling

Static-Priority (or Fixed-Priority) Scheduling

- Different jobs of a task are assigned the same priority.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, i.e., τ_i has higher priority than τ_k if $i < k$.

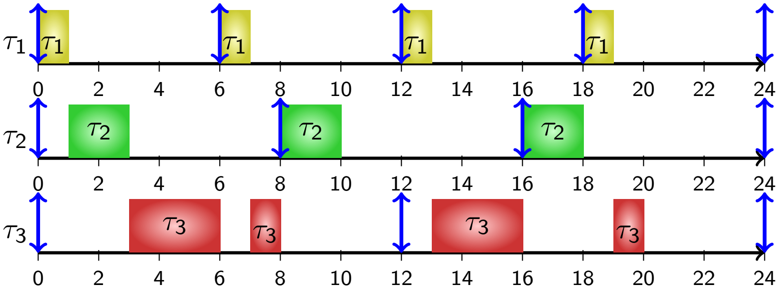
Static-Priority (or Fixed-Priority) Scheduling

- Different jobs of a task are assigned the same priority.
  - Note: we will assume that no two tasks have the same priority.
- We will implicitly index tasks in decreasing priority order, i.e., τ_i has higher priority than τ_k if $i < k$.
- Which strategy is better or the best?
  - largest execution time first?
  - shortest job first?
  - least-utilization first?
  - most importance first?
  - least period first?

Rate-Monotonic (RM) Scheduling (Liu and Layland, 1973)

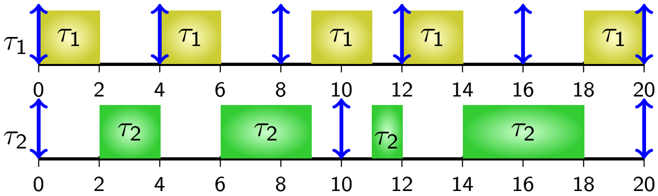
Priority Definition: A task with a smaller period has higher priority, in which ties are broken arbitrarily.

Example Schedule: $\tau_1 = (1, 6, 6)$, $\tau_2 = (2, 8, 8), \tau_3 = (4, 12, 12)$.
$[(C_i, T_i, D_i)]$



Optimality (or not) of RM

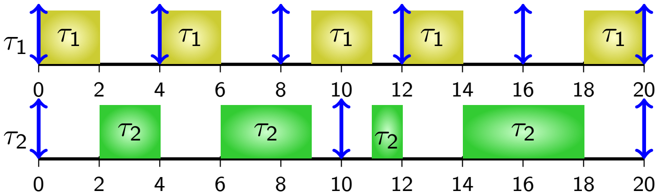
Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



The above system is schedulable.

Optimality (or not) of RM

Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



The above system is schedulable.

No static-priority scheme is optimal for scheduling periodic tasks:

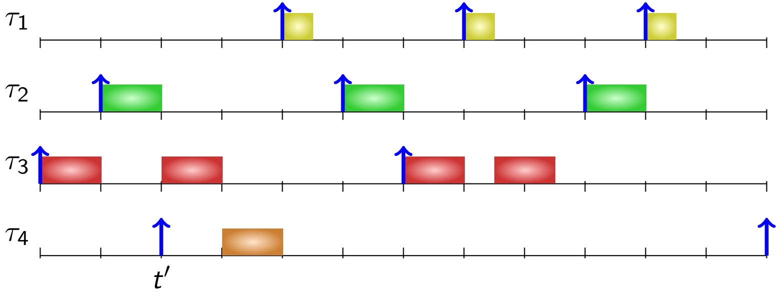
However, a deadline will be missed, regardless of how we choose to (statically) prioritize τ_1 and τ_2.

Corollary

RM is optimal.

Properties of Worst-Case Response Time

Suppose that we are analyzing the worst-case response time of task τ_i. Let us assume that the other $i-1$ higher-priority tasks are already verified to meet their deadlines.



- Suppose t' is the arrival time of a job of task τ_i.
- A higher priority task τ_j may release a job before t' and this job is executed after t'.

Properties of Worst-Case Response Time (cont.)

Let t_j be the arrival time of the *first* job of task τ_j after or at time t'.

- $t_j \geq t'$.
- The remaining execution time of the job of task τ_j arrived before t' and unfinished at time t' is at most C_j.

Since fixed-priority scheduling greedily executes an available job, the system remains busy from t' till the time instant f at which task τ_n finishes the job arrived at time t'. That is,

$$\forall t' < t < f, \qquad C_i + \sum_{j=1}^{i-1} C_j + \sum_{j=1}^{i-1} \max\left\{ \left\lceil \frac{t - t_j}{T_j} \right\rceil C_j, 0 \right\} > t - t'.$$

As a result, ($t - t'$ is replaced by t)

$$\forall 0 < t < f - t', \qquad C_i + \sum_{j=1}^{i-1} C_j + \sum_{j=1}^{i-1} \left\lceil \frac{t - t_j}{T_j} \right\rceil C_j > t.$$

Properties of Worst-Case Response Time (cont.)

The minimum t such that

$$\exists 0 < t \leq T_i, \qquad C_i + \sum_{j=1}^{i-1} C_j + \sum_{j=1}^{i-1} \left\lceil \frac{t - t_j}{T_j} \right\rceil C_j = t.$$

is a safe upper bound on the worst-case response time of task τ_i.

Why do we need to constrain $t \leq T_i$?

Aside: Critical Instants

Definition

The critical instant of a task τ_i is a time instant such that:

1. the job of τ_i released at this instant has the maximum response time of all jobs in τ_i, if the response time of every job of τ_i is at most T_i, the period of τ_i, and
2. the response time of the job released at this instant is greater than T_i if the response time of some job in τ_i exceeds T_i.

Informally, a critical instant of τ_i represents a worst-case scenario from τ_i's standpoint when we use static-priority scheduling.

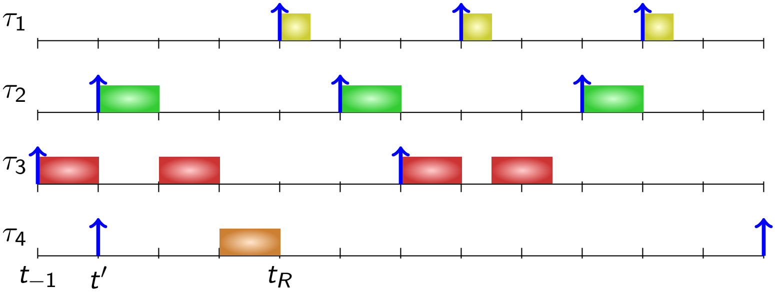
Critical Instants in Static-Priority Systems

Theorem

[Liu and Layland, JACM 1973] The critical instance of task τ_i for a set of independent, preemptable periodic tasks with implicit deadlines is to release the first jobs of all the higher-priority tasks at the same time.

We are not saying that $\tau_1, \ldots, \tau_i$ will all necessarily release their first jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for task τ_i.

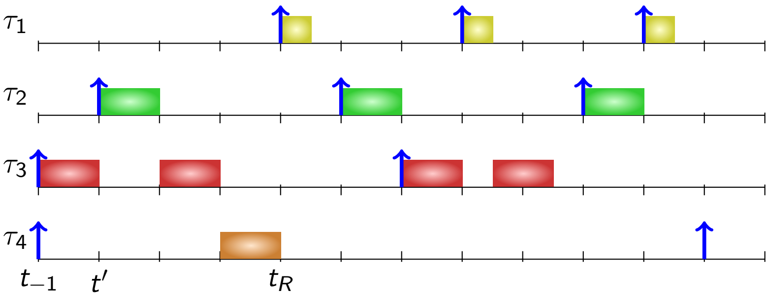
Critical Instants: Informal Proof



Shifting the release time of tasks together will increase the response time of task τ_i.

- Consider a job of τ_i, released at time t', with completion time t_R.
- Let t_{-1} be the latest *idle instant* for $\tau_1, \ldots, \tau_{i-1}$ at or before t_R.
- Let J be τ_i's job released at t'.

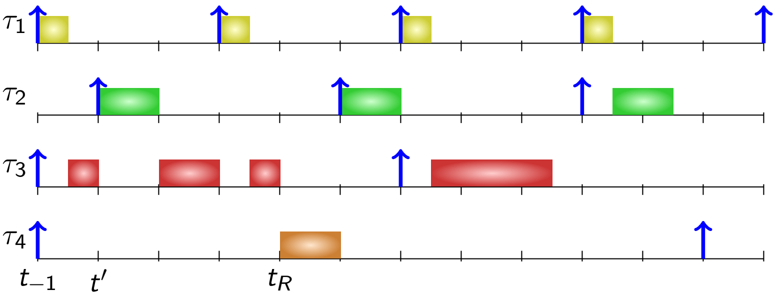
Critical Instants: Informal Proof



We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Moving J from t' to t_{-1} does not decrease the completion time of J.

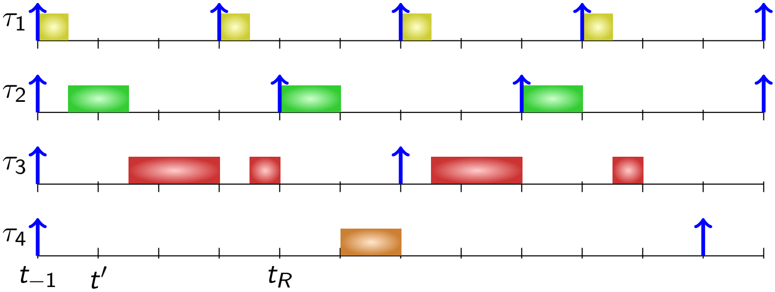
Critical Instants: Informal Proof



We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Releasing τ_1 at t_{-1} does not decrease the completion time of J.

Critical Instants: Informal Proof



We will show that shifting the release time of tasks together will increase the response time of task τ_i.

- Releasing τ_2 at t_{-1} does not decrease the completion time of J.
- Repeating the above movement and proves the criticality of the critical instant

Schedulability Condition

According to the critical instant theorem, to test the schedulability of task τ_i, we have to

1. release all the higher-priority tasks at time 0 together with task τ_i
2. release all the higher-priority task instances as early as they can

We can simply simulate the above behavior to verify whether task τ_i misses the deadline.

TDA (Time-Demand Analysis)

The time-demand function $W_i(t)$ of the task τ_i is defined as follows:

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

Theorem

A system $\mathcal{T}$ of periodic, independent, preemptable tasks is schedulable on one processor by algorithm A if

$$\forall \tau_i \in \mathcal{T} \ \exists t \text{ with } 0 < t \leq D_i \ \ and \ \ W_i(t) \leq t$$

holds. This condition is also necessary for synchronous, periodic task systems and also sporadic task sets.

Note that this holds for implicit-deadline and constrained-deadline task sets. The sufficient condition can be proved by contradiction.

How to Use TDA? (Only for References)

The theorem of TDA might look strong as it requires to check all the time t with $0 < t \leq D_i$ for a given τ_i. There are two ways to avoid this:

- Iterate using $t(k+1) := W_i(t(k))$, starting with $t(0) := \sum_{j=1}^{i} C_j$, and stopping when, for some ℓ, $t(\ell) = W_i(t(\ell))$ or $t(\ell) > D_i$.
- Only consider $t \in \{\ell T_j - \epsilon \mid 1 \leq j \leq i, \ell \in \mathcal{N}^+\}$, where ϵ is a constant close to 0. That is, only consider t at which a job of higher-priority tasks arrives.

Optimality Among Static-Priority Algorithms

Theorem

A system $\mathcal{T}$ of n independent, preemptable, synchronous periodic tasks with implicit deadlines can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.

The proof is omitted. It can be proved by using the critical instant theorem and the TDA analysis.

Harmonic Real-Time Systems

Definition

A system of periodic tasks is said with harmonic periods (also: *simply periodic*) if for every pair of tasks τ_i and τ_k in the system where $T_i < T_k$, T_k is an integer multiple of T_i.

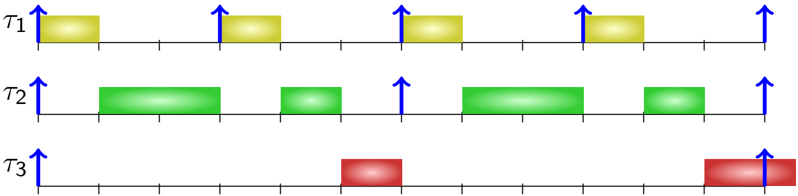
For example: Periods are $2, 6, 12, 24$.

Theorem

[Kuo and Mok]: A system $\mathcal{T}$ of harmonic, independent, preemptable, and implicit-deadline tasks is schedulable on one processor according to the RM algorithm if and only if its total utilization $U(\mathcal{T}) = \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}$ is less than or equal to 1.

Proof for Harmonic Systems

The case for the "only-if" part is skipped.



By using the contrapositive proof approach, suppose that $\mathcal{T}$ is not schedulable and τ_i misses its deadline. We will prove that the utilization must be larger than 1.

- The response time of τ_i is larger than D_i.
- By critical instants, releasing all the tasks $\tau_1, \tau_2, \ldots, \tau_i$ at time 0 will lead to a response time of τ_i larger than D_i.

Proof for Harmonic Systems (cont.)

As the schedule is workload-conserving, we know that from time 0 to time D_i, the whole system is executing jobs. Therefore,

$$D_i < \text{the workload released in time interval } [0, D_i)$$

$$= \sum_{j=1}^{i} C_j \cdot (\text{ the number of job releases of } \tau_j \text{ in time interval } [0, D_i))$$

$$= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{D_i}{T_j} \right\rceil =^* \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},$$

where $=^*$ is because *$D_i = T_i$ is an integer multiple of T_j when $j \leq i$*.

Proof for Harmonic Systems (cont.)

As the schedule is workload-conserving, we know that from time 0 to time D_i, the whole system is executing jobs. Therefore,

$$
\begin{aligned}
D_i &< \text{the workload released in time interval } [0, D_i) \\
&= \sum_{j=1}^{i} C_j \cdot (\text{ the number of job releases of } \tau_j \text{ in time interval } [0, D_i)) \\
&= \sum_{j=1}^{i} C_j \cdot \left\lceil \frac{D_i}{T_j} \right\rceil =^* \sum_{j=1}^{i} C_j \cdot \frac{D_i}{T_j},
\end{aligned}
$$

where $=^*$ is because *$D_i = T_i$ is an integer multiple of T_j when $j \leq i$*.

By canceling D_i, we reach the contradiction by having

$$1 < \sum_{j=1}^{i} \frac{C_j}{T_j} \leq \sum_{\tau_j \in \mathcal{T}} \frac{C_j}{T_j}.$$

Utilization-Based Schedulability Test

- Task utilization:

$$U_i := \frac{C_i}{T_i}.$$

- System (total) utilization:

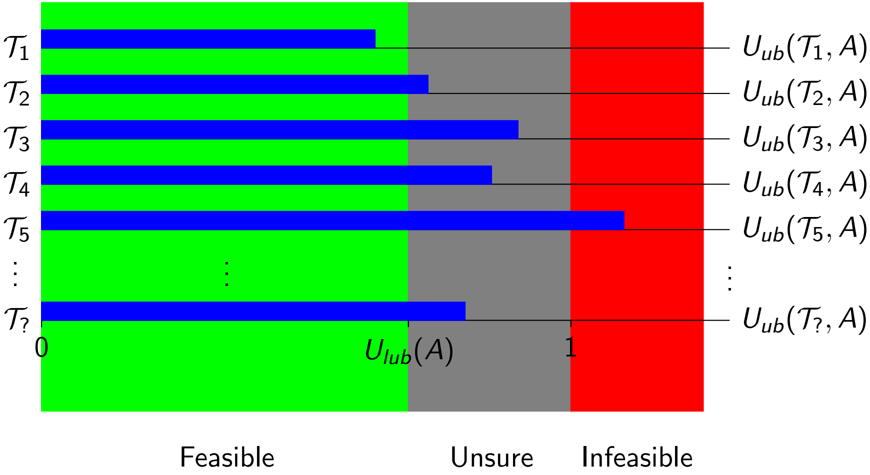
$$U(\mathcal{T}) := \sum_{\tau_i \in \mathcal{T}} \frac{C_i}{T_i}.$$

A task system $\mathcal{T}$ fully utilizes the processor under scheduling algorithm A if any increase in execution time (of any task) causes A to miss a deadline. In this case, $U(\mathcal{T})$ is an upper bound on utilization for A, denoted $U_{ub}(\mathcal{T}, A)$.

$U_{lub}(A)$ is the least upper bound for algorithm A:

$$U_{lub}(A) = \min_{\mathcal{T}} U_{ub}(\mathcal{T}, A)$$

What is $U_{lub}(A)$ for?



Liu and Layland Bound

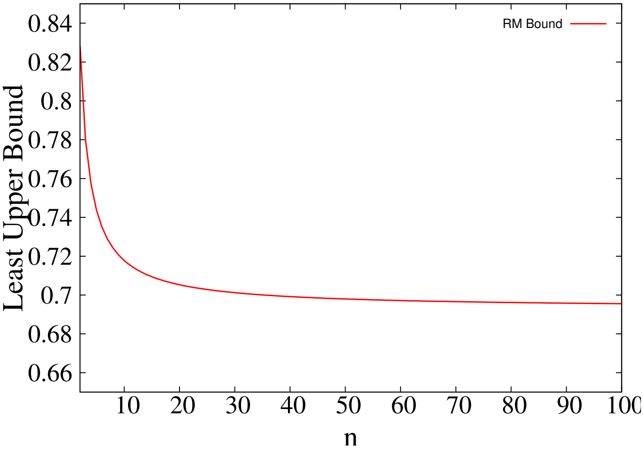
Theorem

[Liu and Layland] A set of n independent, preemptable periodic tasks with implicit deadlines can be scheduled on a processor according to the RM algorithm if its total utilization U is at most $n(2^{\frac{1}{n}} - 1)$. In other words,

$$U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq 0.693.$$

n	$U_{lub}(RM, n)$	n	$U_{lub}(RM, n)$
2	0.828	3	0.779
4	0.756	5	0.743
6	0.734	7	0.728
8	0.724	9	0.720
10	0.717	∞	$0.693 = ln2$

Least Upper Bound



Comparison between RM and EDF (Implicit Deadlines)

RM

- Low run-time overhead: $O(1)$ with priority sorting in advance
- Optimal for static-priority
- Response time analysis is $\mathcal{N}P$-hard (even if the relative deadline $=$ period)
- Least upper bound: 0.693
- In general, more preemption

EDF

- High run-time overhead: $O(\log n)$ with balanced binary tree
- Optimal for dynamic-priority
- Schedulability test is easy (when the relative deadline $=$ period)
- Least upper bound: 1
- In general, less preemption