Schedulability of Sporadic Tasks on Uniprocessor Systems

Prof. Dr. Jian-Jia Chen

LS 12, TU Dortmund

Recurrent Task Models (Revisited)

- 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 of task τ_i, indicating the length of the maximum interval before a job must be finished
  - (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.

Recurrent Task Models (Revisited)

- 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 of task τ_i, indicating the length of the maximum interval before a job must be finished
  - (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
  - D_i is the relative deadline for each job of 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.

Relative Deadline <=> Period (Revisit)

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.

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: $U(\mathbf{T}) := \sum_{\tau_i \in \mathbf{T}} U_i$.

Static-Priority Scheduling

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

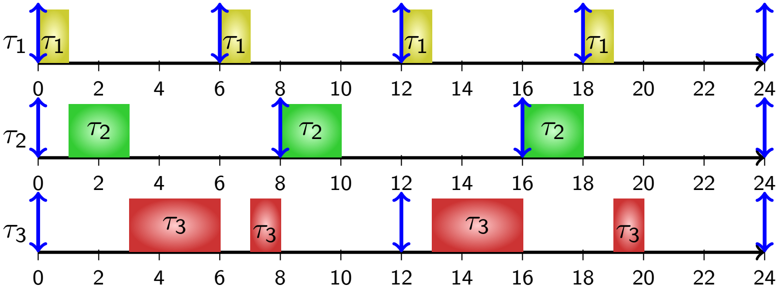
Static-Priority Scheduling

- Different jobs of a task are assigned the same priority.
  - Note: we will assume that no two tasks have the same priority. (Why?)
- 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: $\tau_1 = (1, 6, 6)$, $\tau_2 = (2, 8, 8), \tau_3 = (4, 12, 12)$. $[(C_i, T_i, D_i)]$

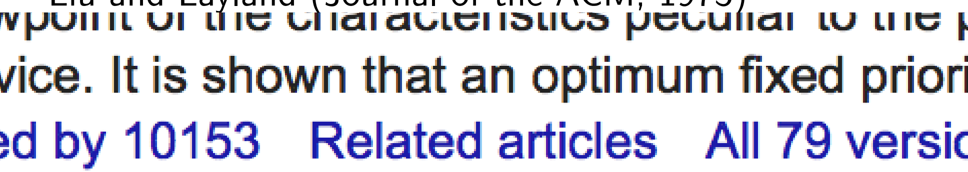


Liu and Layland (Journal of the ACM, 1973)

Scheduling algorithms for multiprogramming in a hard-real-time environment
CL Liu, JW Layland - Journal of the ACM (JACM), 1973 - dl.acm.org
Abstract The problem of multiprogram scheduling on a single processor is studied from the viewpoint of the characteristics peculiar to the program functions that need guaranteed service. It is shown that an optimum fixed priority scheduler possesses an upper bound to ...
Cited by 10153   Related articles   All 79 versions   Web of Science: 2251   Cite   Save

googled on 16,02,2016

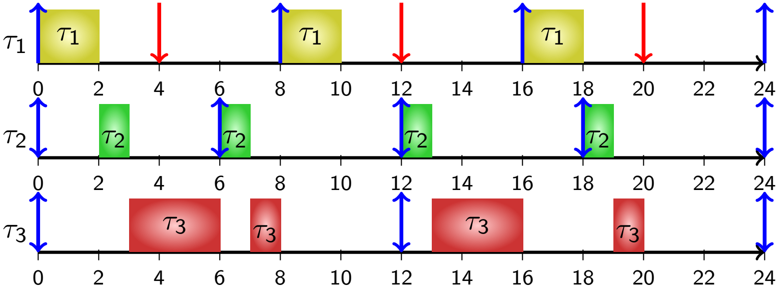
Liu and Layland (Journal of the ACM, 1973)



Deadline-Monotonic (DM) Scheduling (Leung and Whitehead)

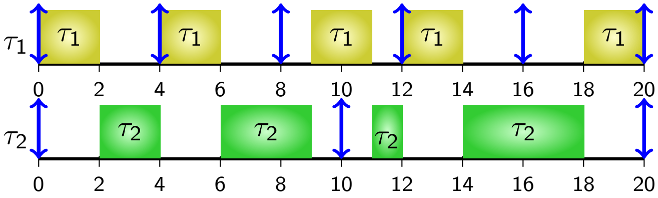
Priority Definition: A task with a smaller relative deadline has higher priority, in which ties are broken arbitrarily.

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



Optimality (or not) of RM and DM

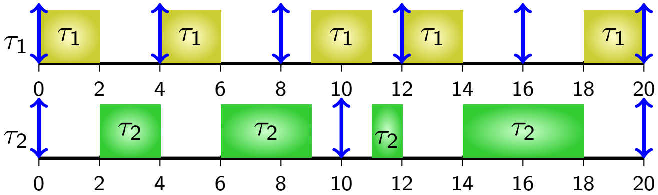
Example Schedule: $\tau_1 = (2, 4, 4)$, $\tau_2 = (5, 10, 10)$



No static-priority scheme is optimal for scheduling periodic tasks:
The above system is schedulable.

Optimality (or not) of RM and DM

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



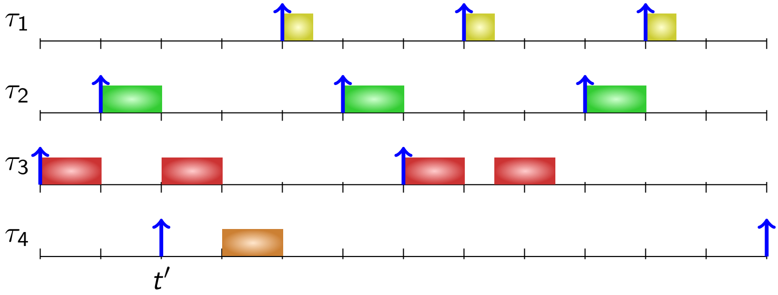
No static-priority scheme is optimal for scheduling periodic tasks:
The above system is schedulable.
However, a deadline will be missed, regardless of how we choose to (statically) prioritize τ_1 and τ_2.

Corollary

Neither RM nor DM is optimal.

Worst-Case Response Time (Constrained-Deadline)

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



- Suppose t' is the arrival time of a job of task τ_k.
- 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_k finishes the job arrived at time t'. That is,

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

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

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

Properties of Worst-Case Response Time (cont.)

The minimum $0 < t \leq D_k$ such that

$$C_k + \sum_{j=1}^{k-1} C_j + \sum_{j=1}^{k-1} \left\lceil \frac{t}{T_j} \right\rceil C_j = t.$$

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

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

Critical Instants in Static-Priority Systems

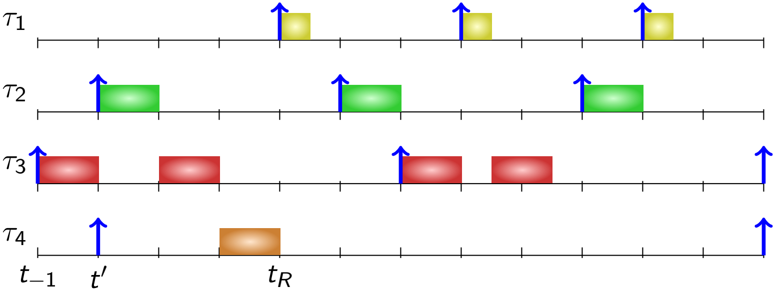
Theorem

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

We are not saying that $\tau_1, \ldots, \tau_k$ 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_k.

This argument also works for task sets with constrained deadlines.

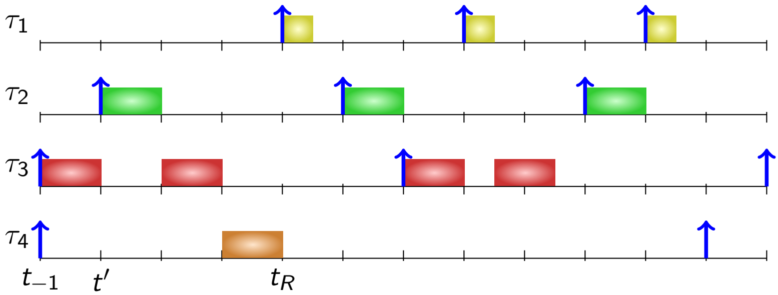
Critical Instants: Informal Proof



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

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

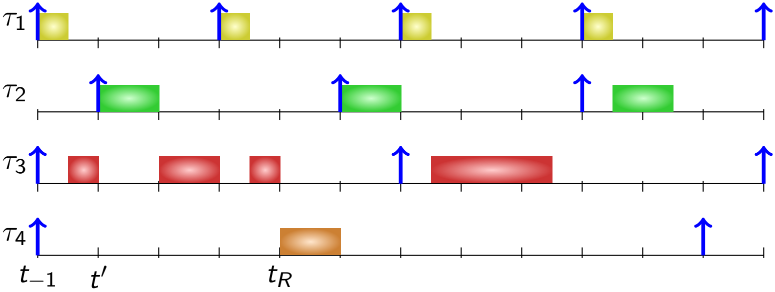
Critical Instants: Informal Proof



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

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

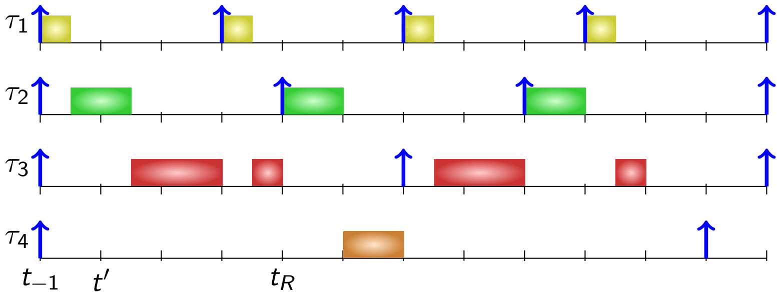
Critical Instants: Informal Proof



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

- 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_k.

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

Outline

Schedulability for Static-Priority Scheduling

- Demand-Based Analysis
- Optimality of RM
- Utilization-Based Analysis (Relative Deadline = Period)
- Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling

Necessary/Sufficient Schedulability Test

- The issue for timing analysis is on how to analyze the schedulability.
  - Sufficient Test: If A holds, then the task set is schedulable (by EDF, RM, or DM).
  - Necessary Test: If the task set is schedulable by EDF (or RM/DM), then B holds.
  - Exact Test: The task set is schedulable by EDF (or RM/DM) if and only if A^* holds.

Necessary and Sufficient (Exact) RM-Schedulability

- Time-demand analysis (TDA) was proposed by Lehoczky, Sha, and Ding [RTSS 1989].
- TDA can be applied to produce a schedulability test for any fixed-priority algorithm that ensures that each job of every task completes before the next job of that task is released.
- For some important task models and scheduling algorithms, this schedulability test will be necessary and sufficient.

Schedulability Condition

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

1. release all the higher-priority tasks at time 0 together with task τ_k
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_k misses the deadline.

Several examples are used in Slide 6/8/9 to demonstrate this behavior.

Schedulability Test

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

$$W_k(t) = C_k + \sum_{j=1}^{k-1} \left\lceil \frac{t}{T_j} \right\rceil C_j.$$

Theorem

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

$$\forall \tau_k \in \mathbf{T}\ \exists t \text{ with } 0 < t \leq D_k \ \ \textit{and} \ \ W_k(t) \leq t$$

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?

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

- Iterate using $t(\ell + 1) := W_k(t(\ell))$, starting with $t(0) := \sum_{j=1}^{k} C_j$ and stopping, when $t(\ell) = W_k(t(\ell))$ or $t(\ell) > D_i$ for some ℓ.
- 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.

Complexity of TDA Analysis

The complexity to analyze weather a task τ_k can meet the timing constraint is $O(kD_k)$.

- $O(kD_k)$ has polynomial time complexity, if the input is in the unary format, i.e. if D_k is 6, the input is 111111 instead of the binary 110.
- It has exponential runtime for input in the binary format.
- Formally, this is called with pseudo-polynomial time complexity.

Theorem

Eisenbrand and Rothvoss [RTSS 2008]: Fixed-Priority Real-Time Scheduling: Response Time Computation Is $\mathcal{N}P$-Hard

Optimality Among Static-Priority Algorithms

Theorem

A system **T** of independent, preemptable, synchronous periodic tasks that have relative deadlines equal to their respective periods can be feasibly scheduled on one processor according to the RM algorithm whenever it can be feasibly scheduled according to any static priority algorithm.

We will only discuss systems with 2 tasks, and the generalization is left as an exercise.

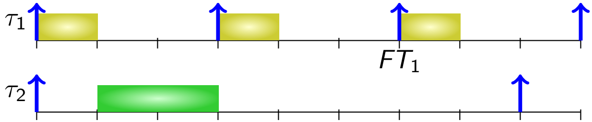
- Suppose that $T_1 = D_1 < D_2 = T_2$ and τ_2 has the higher priority.
- We would like to swap the priorities of τ_1 and τ_2.
- Without loss of generality, the response time of τ_1 after priority swapping is always equal to (or no more than) C_1.
- By the critical instant theorem, we only need to check response time of the first job of τ_2 during a critical instant.
- Assuming that non-RM priority ordering is schedulable, the critical instant theorem also implies that $C_1 + C_2 \leq T_1$.

Optimality Among Static-Priority Algorithms (cont.)

After swapping (τ_1 has higher priority), there are two cases:

Case 1

There is sufficient time to complete all $F+1$ jobs of τ_1 before the second job arrival of τ_2, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$. In other words, $C_1 + F \cdot T_1 < T_2$.



To be schedulable

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

By $C_1 + C_2 \leq T_1$, we have

$$F(C_1 + C_2) \leq F \cdot T_1$$

$$\stackrel{F \geq 1}{\Rightarrow} FC_1 + C_2 \leq F \cdot T_1$$

$$(F+1)C_1 + C_2 \leq F \cdot T_1 + C_1$$

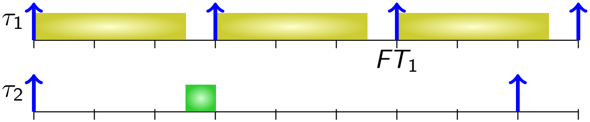
$$\Rightarrow (F+1)C_1 + C_2 < T_2$$

Optimality Among Static-Priority Algorithms (cont.)

After swapping (τ_1 has higher priority), there are two cases:

Case 2

The $(F+1)$-th job of τ_1 does not complete before the arrival of the second job of τ_2. In other words, $C_1 + F \cdot T_1 \geq T_2$, where $F = \left\lfloor \frac{T_2}{T_1} \right\rfloor$.



To be schedulable

$FC_1 + C_2 \leq FT_1$ must hold.

By $C_1 + C_2 \leq T_1$, we have

$$F(C_1 + C_2) \leq F \cdot T_1$$

$$\overset{F \geq 1}{\Rightarrow} FC_1 + C_2 \leq F \cdot T_1 \leq T_2$$

Remarks on the Optimality

We have shown that if any synchronous periodic two-task system with implicit deadlines ($D_i = T_i$) is schedulable according to arbitrary fixed-priority assignment, then it is also schedulable according to RM.

Exercise: Complete proof by extending argument to n synchronous periodic tasks.

Note: When $D_i \leq T_i$ for all tasks, DM (Deadline Monotonic) can be shown to be an optimal static-priority algorithm for all synchronous task sets using a similar argument. The proof is also left as an exercise.

Outline

Schedulability for Static-Priority Scheduling

- Demand-Based Analysis
- Optimality of RM
- Utilization-Based Analysis (Relative Deadline = Period)
- Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling

Definitions

- Task utilization:

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

- System (total) utilization:

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

Harmonic Real-Time Systems

Definition

A system of periodic tasks is called 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.

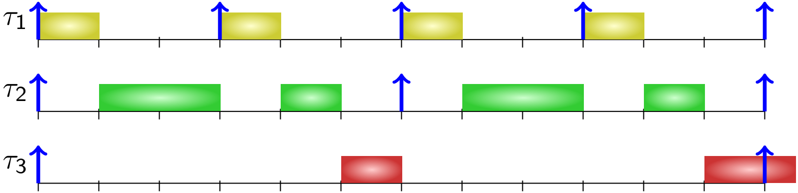
For example: Periods are $2, 6, 12, 24$.

Theorem

[Kuo and Mok]: A system $\mathbf{T}$ 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 $U(\mathbf{T}) = \sum_{\tau_j \in \mathbf{T}} \frac{C_j}{T_j}$ 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 **T** is not schedulable and τ_k misses its deadline. We will prove that the utilization must be larger than 1.

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

Proof for Harmonic Systems (cont.)

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

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

where $=^*$ is because *$D_k = T_k$ is an integer multiple of T_j when $j \leq k$*.

Proof for Harmonic Systems (cont.)

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

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

where $=^*$ is because *$D_k = T_k$ is an integer multiple of T_j when $j \leq k$*.

By canceling D_k, we reach the contradiction by having

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

Utilization-Based Schedulability Test

- Task utilization:

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

- System (total) utilization:

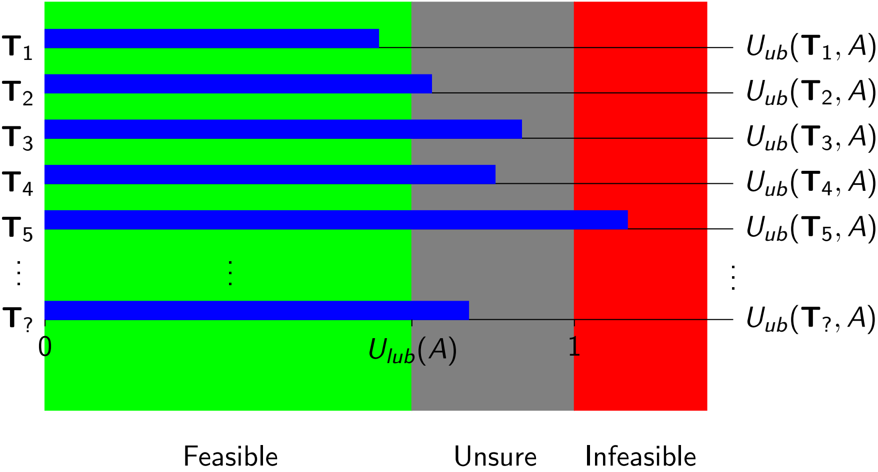
$$U(\mathbf{T}) := \sum_{\tau_i \in \mathbf{T}} \frac{C_i}{T_i}.$$

A task system $\mathbf{T}$ 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, $U(\mathbf{T})$ is an upper bound on utilization for A, denoted $U_{ub}(\mathbf{T}, A)$.

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

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

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



Liu and Layland Bound

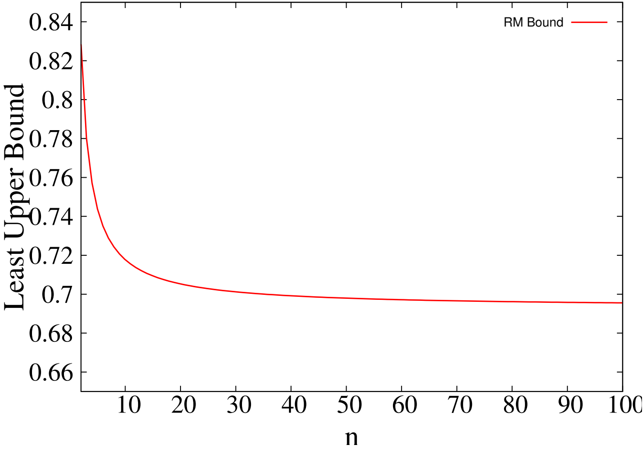
Theorem

[Liu and Layland] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods 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	∞	$0.693 = \ln(2)$

Least Upper Bound



Proof Sketch for $U_{ulb}(RM, n)$

Note: The original proof for this theorem by Liu and Layland is not correct. For a corrected proof, see R. Devillers & J. Goossens at `http://dev.ulb.ac.be/sched/articles/lub.ps`. Note the proof presented here is VERY different from the others, including the one from Buttazzo's textbook.

1. We will start from the exact test and analyze the schedulability under RM of task τ_n.
2. This will easily lead us to consider only the special case where $T_n \leq 2T_1$.
3. We will then show the schedulability condition of τ_n under RM
4. The least utilization bound is then derived based on the above schedulability condition.

Utilization Bound Proof: Step 1

Theorem

[Bini and Buttazzo, ECRTS 2001] A system of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if

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

Suppose that task τ_n is not schedulable under RM. We will prove that $\Pi_{i=1}^{n}(U_i + 1) > 2$.

Worst-Case: $T_n \leq 2T_1$

By the exact test, we know that for all $0 < t \leq T_n$

$$W_n(t) = C_n + \sum_{i=1}^{n-1} \left\lceil \frac{t}{T_i} \right\rceil C_i > t.$$

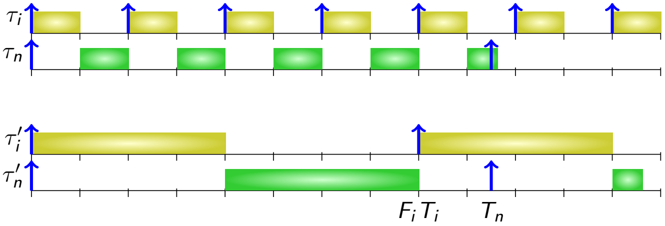
Now, suppose again F_i is $\left\lfloor \frac{T_n}{T_i} \right\rfloor$. For all $0 < t \leq T_n$

$$\left\lceil \frac{t}{T_i} \right\rceil C_i \leq \left\lceil \frac{t}{F_i T_i} \right\rceil F_i C_i$$

Therefore, by changing the period of task τ_i to $F_i T_i$ and the execution time from C_i to $F_i C_i$, task τ_n in the new task set remains unschedulable under RM.

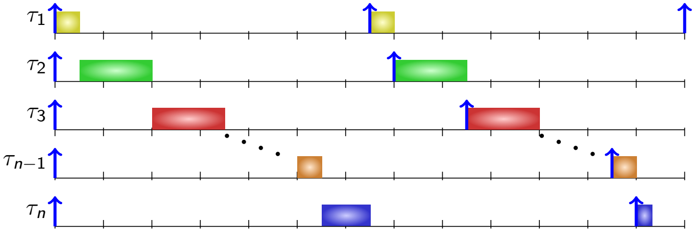
After changing the periods, we reorder the tasks according to their new periods. Does this affect the non-schedulability of task τ_n?

$T_n \leq 2T_1$ (cont.)



Hyperbolic Bound: Structure

For the rest of the proof, we only consider $T_n \leq 2T_1$. The non-schedulability also implies the following structure:



$$C_n + \sum_{j=1}^{n-1} C_j + \sum_{j=0}^{i-1} C_j > T_i, \forall i = 1, 2, \ldots, n-1,$$

$$C_n + 2\sum_{j=1}^{n-1} C_j > T_n,$$

where C_0 is defined as 0 for brevity.

Hyperbolic Bound: Structure

This means, C_n must be *sufficiently large* to enforce the above conditions.

Let's now recall what we were doing:

- We were given a set of tasks, in which the utilization U_i of each task τ_i is given.
- We wanted to prove that the task set is always schedulable under RM no matter how the periods are assigned under certain utilization constraints.
- What we have done so far is using the contraposition that there exists at least one assignment of periods to make task τ_n not schedulable under RM, when the utilization is larger than a value given by certain conditions.

Hyperbolic Bound: U_n

For a critical value of U_n, if we reduce U_n by a small value, the above non-schedulability condition will not be satisfied any more. So, the critical value is equivalent to the minimum U_n to enforce the following condition:

$$C_n + \sum_{j=1}^{n-1} C_j + \sum_{j=0}^{i-1} C_j \geq T_i \geq 0, \forall i = 1, 2, \ldots, n-1,$$

$$C_n + 2\sum_{j=1}^{n-1} C_j = T_n,$$

In fact, we can also normalize T_n to 1. The above condition to get the minimum U_n is equivalent to the following linear programming

$$\text{minimize } C_n = T_n - 2\sum_{j=1}^{n-1} U_j T_j$$

$$\text{s.t.} T_n - 2\sum_{j=1}^{n-1} U_j T_j + \sum_{j=1}^{n-1} U_j T_j + \sum_{j=0}^{i-1} U_j T_j \geq T_i \geq 0, \forall i = 1, \ldots, n-1$$

Extreme Point Theory in Linear Programming

$$\text{minimize } C_n = T_n - 2\sum_{j=1}^{n-1} U_j T_j$$

$$\text{s.t. } T_n - \sum_{j=i}^{n-1} U_j T_j \geq T_i \geq 0, \forall i = 1, \ldots, n-1$$

The optimal solution of the above linear programming is achieved when $T_i > 0$ and all the other $n-1$ linear constraints are with $=$ instead of $\geq$ by the extreme point theory. (details omitted here and to be discussed later in the lecture.) That is, the minimum U_n is achieved when

$$T_i = T_n - \sum_{j=i}^{n-1} U_j T_j, \forall i = 1, \ldots, n-1$$

$$C_i = T_{i+1} - T_i, \forall i = 1, \ldots, n-1$$

Hyperbolic Bound: Final

$$U_i = \frac{C_i}{T_i} = \frac{T_{i+1} - T_i}{T_i} = \frac{T_{i+1}}{T_i} - 1, \forall i = 1, \ldots, n-1$$

$$C_n^* = 2T_1 - T_n \Rightarrow U_n^* = 2\frac{T_1}{T_n} - 1,$$

where C_n^* is the optimal solution of the above linear programming. The non-schedulability of task τ_n implies that

$$\begin{aligned} U_n > U_n^* &= 2\frac{T_1}{T_n} - 1 = 2\left(\frac{T_1}{T_2}\frac{T_2}{T_3}\cdots\frac{T_{n-1}}{T_n}\right) - 1 \\ &= 2\frac{1}{\Pi_{i=1}^{n-1}(U_i + 1)} - 1 \\ \Rightarrow & \Pi_{i=1}^{n}(U_i + 1) > 2. \end{aligned}$$

Recall Utilization Bound Proof: Step 1

Theorem

[Bini and Buttazzo, ECRTS 2001] A system of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be scheduled on a processor according to the RM algorithm if

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

Utilization Bound Proof: Step 2 Calculate $U_{lub}(RM, n)$

- So what is the minimum (infmum) $\sum_{i=1}^{n} U_i$ to enforce $\Pi_{i=1}^{n}(U_i + 1) > 2$? (see A-Mathmatics.pdf)
  - It should be clear that the infmum $\sum_{i=1}^{n} U_i$ happens when $\Pi_{i=1}^{n}(U_i + 1) = 2$, and $U_i = 2^{\frac{1}{n}} - 1$.
  - $U_{lub}(RM, n) = n(2^{\frac{1}{n}} - 1) \geq \lim_{n\to\infty} n(2^{\frac{1}{n}} - 1) = \ln(2)$.

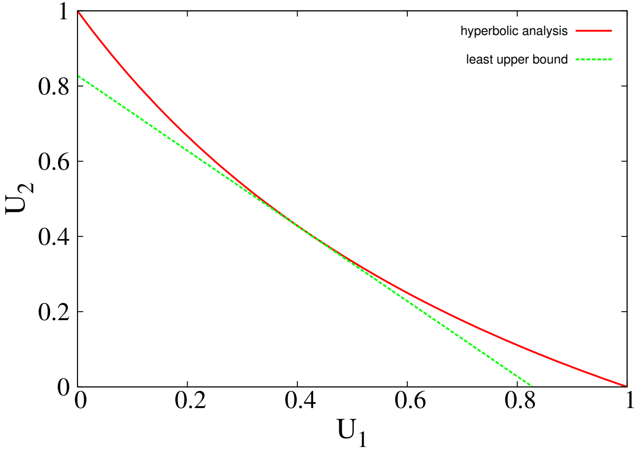
This concludes the proof of the following theorem:

Theorem

[Liu and Layland, JACM 1973] A set of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods 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,

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

Least Upper Bound



On-Site Exercise: Is This Schedulable under RM?

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	7	14	26	26	79	292
D_i	2	6	13	25	26	77	291

On-Site Exercise: Is This Schedulable under RM?

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	7	14	26	26	79	292
D_i	2	6	13	25	26	77	291

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	6	13	25	26	77	291
D_i	2	6	13	25	26	77	291

On-Site Exercise: Is This Schedulable under RM?

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	7	14	26	26	79	292
D_i	2	6	13	25	26	77	291

C_i	0.2	2	2	1.5	1	14	28.8
T_i	2	6	13	25	26	77	291
D_i	2	6	13	25	26	77	291

C_i	0.2	2	2	1.5	1	14	28.8
T'_i	2	6	12	24	24	72	288
D'_i	2	6	12	24	24	72	288
U'_i	0.1	1/3	1/6	0.0625	0.0417	0.195	0.1

Remarks on Harmonic Task Set

- Now, we know that if the total utilization is larger than 0.693, the utilization-bound schedulability cannot provide guarantees for schedulability or unschedulability.
- Sometimes, we can manipulate the periods such that the new task set is a harmonic task set and its schedulability can be used.

Outline

Schedulability for Static-Priority Scheduling

- Demand-Based Analysis
- Optimality of RM
- Utilization-Based Analysis (Relative Deadline = Period)
- Arbitrary Deadlines

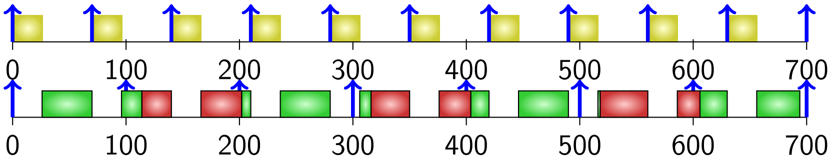
Schedulability for Dynamic-Priority Scheduling

TDA for Arbitrary Deadlines (Details are in Appendix)

- The TDA scheduling condition is valid only if each job of every task completes before the next job of that task is released.
- We now consider a schedulability check in which tasks may have *relative deadlines larger than their periods*.
  - Note: In this model, a task may have multiple ready jobs. We assume they are scheduled on a First-Come-First Serve (FCFS) basis.

Straightforward Analysis for Arbitrary Deadlines

The worst-case response time of τ_i by only considering the first job of τ_i at the critical instant is too optimistic when the relative deadline of τ_i is larger than the period.



Consider two tasks:

- τ_1 has period 70 and execution time 26 and τ_2 is with period 100 and execution time 62.
- τ_2's seven jobs have the following response times, respectively: 114, 102, 116, 104, 118, 106, 94.
- Note that the first job's response time is not the longest.

Priority Ordering for Tasks with Arbitrary-Deadline

- There is no greedy strategy for optimal ordering
  - DM or RM is not an optimal static-priority scheme any more.
- Audsley's approach (1991):
  - Use TDA to find the worst-case response time of task τ_i by assuming the others have higher priority
  - Among those tasks whose worst-case response times are less than or equal to the relative deadlines, choose one of them as the lowest-priority task
  - Reduce the problem by removing this lowest-priority task and repeat the above procedure
- Audsley's approach is optimal.

Outline

Schedulability for Static-Priority Scheduling

- Demand-Based Analysis
- Optimality of RM
- Utilization-Based Analysis (Relative Deadline = Period)
- Arbitrary Deadlines

Schedulability for Dynamic-Priority Scheduling

Utilization-Based Test for EDF Scheduling

Theorem

Liu and Layland: A task set **T** of independent, preemptable, periodic tasks with relative deadlines equal to their periods can be feasibly scheduled (under EDF) on one processor if and only if its total utilization U is at most one.

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.
- We prove the contrapositive, i.e., if **T** is not schedulable, then $U > 1$.
  - Let $J_{i,k}$ be the first job to miss its absolute deadline at $d_{i,k}$.
  - Let t_{-1} be the last idle instant or a job with absolute deadline $> d_{i,k}$ is executed before $d_{i,k}$.
  - t_{-1} could be 0 if there is no idle time.

(cont.)

Proof of Utilization-Bound Test for EDF

Proof.

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

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

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

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 **T** of independent, preemptable, periodic tasks with relative deadlines equal to or less than their periods can be feasibly scheduled (under EDF) on one processor if

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

Note: This theorem only gives a sufficient condition.

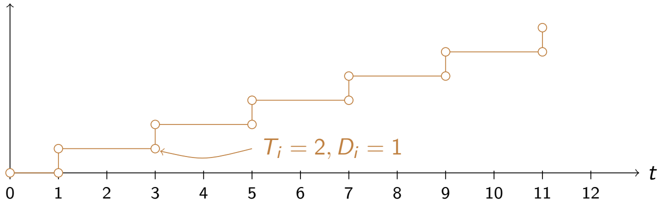
Necessary and Sufficient Conditions

Theorem

Define demand bound function $dbf(\tau_i, t)$ as

$$dbf(\tau_i, t) = \max\left\{0, \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor\right\} C_i = \max\left\{0, \left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1\right\} C_i.$$

A task set $\mathbf{T}$ of independent, preemptable, periodic tasks can be feasibly scheduled (under EDF) on one processor if and only if $\forall\ L \geq 0,\ \sum_{i=1}^{n} dbf(\tau_i, L) \leq L$.



Proof for EDF Schedulability Test

- The processor demand in time interval $[t_1, t_2]$ is the computation demand that must be finished in interval $[t_1, t_2]$. That is, only jobs that arrive no earlier than t_1 and have absolute deadline no more than t_2 are considered.
- The processor demand $g_i([t_1, t_2])$ contributed by task τ_i is

$$g_i([t_1, t_2]) = C_i \cdot \max\left\{0, \underbrace{\left\lfloor \frac{t_2 + T_i - D_i - \phi_i}{T_i} \right\rfloor}_{\text{\# of jobs with deadline no more than } t_2} - \underbrace{\left\lceil \frac{t_1 - \phi_i}{T_i} \right\rceil}_{\text{\# of jobs with arrival time less than } t_1} \right\}$$

- The feasibility is guaranteed if and only if in any interval $[t_1, t_2]$, the processor demand is no more than the available time, i.e.,

$$t_2 - t_1 \geq \sum_{i=1}^{n} g_i(t_1, t_2) \geq \sum_{i=1}^{n} \left\lfloor \frac{t_2 + T_i - D_i - t_1}{T_i} \right\rfloor$$

- Replacing $t_2 - t_1$ by L, we conclude the proof.

Complexity of the Exact Analysis

For analyzing whether a task set can be schedulable by EDF, the time complexity is $O(nL_{\max})$, where $L_{\max}$ is the hyper-period $LCM(T_1, T_2, \ldots, T_n)$.

- It takes exponential-polynomial time (not pseudo-polynomial time). Why?

Theorem

Ekberg and Wang [ECRTS 2015]: testing EDF schedulability of such a task set is (strongly) co$\mathcal{NP}$-hard. That is, deciding whether a task set is not schedulable by EDF is (strongly) $\mathcal{NP}$-hard.

Comparison between RM and EDF (Implicit Deadlines)

RM

- Low run-time overhead: $O(1)$ with priority sorting in advance
- Optimal for static-priority (need: synchronous)
- Schedulability test is $\mathcal{NP}$-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

Appendix: TDA for Arbitrary Deadlines

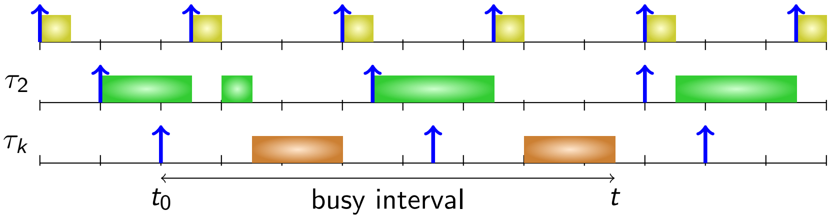
Busy Intervals

Definition

A τ_k-level busy interval $(t_0, t]$ of task τ_k begins at an instant t_0 when

1. all jobs in τ_k released before t have completed, and
2. a job of τ_k releases.

The interval ends at the first instant t after t_0 when all jobs in τ_k released since t_0 are complete.



TDA Analysis (sketched)

Theorem

We are given a set $\mathbf{T}$ of sporadic, independent, preemptable tasks.

1. If $\forall \tau_k \in \mathbf{T}$ $\exists t$ with $0 < t \leq \min\{T_k, D_k\}$ and $W_k(t) \leq t$, then $\mathbf{T}$ is schedulable on one processor by algorithm A for priority ordering.
2. Otherwise, we have to solve the following equation iteratively

$$t^{(\ell+1)} = \sum_{j=1}^{k} \left\lceil \frac{t^{(\ell)}}{T_j} \right\rceil C_j,$$

with initialization $t^{(0)} = \sum_{j=1}^{k} C_j$. If the *maximum response time* of the jobs of τ_k released in time $(0, t]$ is less than the relative deadline, $\mathbf{T}$ is schedulable; otherwise $\mathbf{T}$ is not schedulable.

Response Times

Lemma

The maximum response time $W_{k,j}$ of the j-th job of τ_k in an in-phase τ_k busy period is equal to the smallest value of t that satisfies the equation

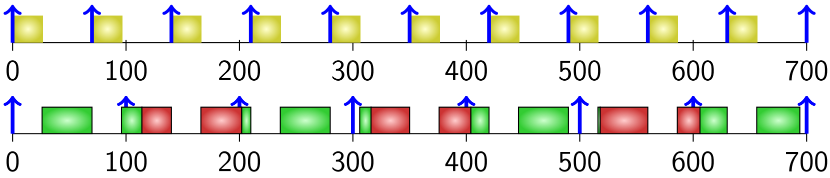
$$t = w_{k,j}(t + (j-1) \cdot T_k) - (j-1) \cdot T_k,$$

where $w_{k,j}(t) = jC_k + \sum_{i=1}^{k-1} \left\lceil \frac{t}{T_i} \right\rceil C_i$.

This should be clear now.

An Example TDA Analysis

Suppose that D_2 is 120 for this example.



$$t = w_{2,1}(t) = C_2 + \sum_{i=1}^{1} \left\lceil \frac{t}{T_i} \right\rceil C_i = 62 + \left\lceil \frac{t}{70} \right\rceil 26$$

$\rightarrow W_{2,1} = 114$

$$t = w_{2,2}(t + T_2) - T_2 = 124 + \left\lceil \frac{t+100}{70} \right\rceil 26 - 100$$

$\rightarrow W_{2,2} = 102$

$$t = w_{2,3}(t + 2T_2) - 2T_2 = 186 + \left\lceil \frac{t+200}{70} \right\rceil 26 - 200$$

$\rightarrow W_{2,3} = 116$

Correctness of the TDA for Arbitrary Relative Deadlines

Lemma

The response time $W_{k,j}$ of the j-th job of τ_k executed in an in-phase τ_k busy interval is no less than the response time of the j-th job of τ_k executed in any τ_k busy interval.

Lemma

The number of jobs in τ_k that are executed in an in-phase τ_k busy interval is never less than the number of jobs in this task that are executed in a τ_k busy interval of arbitrary phase.