CSCE 990: *Real-Time Systems*

Static-Priority Scheduling

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/RealTimeSystems

Static-priority Scheduling

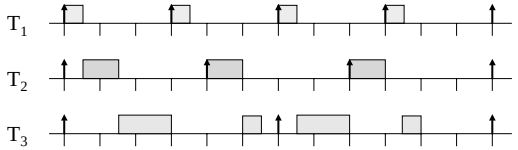
- We now consider **static-priority scheduling**.
  - Under static-priority scheduling, different jobs of a task are assigned the same priority.
  - We will assume that tasks are indexed in decreasing priority order, i.e., T_i has higher priority than T_k if $i < k$.
  - **Notation:**
    - π_i denotes the priority of T_i.
    - $\mathbf{T}_i$ denotes the subset of tasks with equal or higher priority than T_i.
    - **Note:** In some of the papers we will read, it is assumed no two tasks have the same priority. (Is this OK?)

Rate-monotonic Scheduling

(Liu and Layland)

Priority Definition: Tasks with smaller periods have higher priority.

Example Schedule: Three tasks, $T_1 = (3,0.5)$, $T_2 = (4,1)$, $T_3 = (6,2)$.



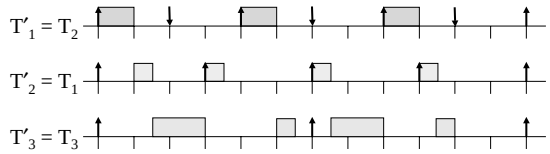
Deadline-monotonic Scheduling

(Leung and Whitehead)

Priority Definition: Tasks with smaller relative deadlines have higher priority.

Same as rate-monotonic if each task's relative deadline equals its period.

Example Schedule: Let's change the RM example by giving T_2 a tighter deadline: $T_1 = (3,0.5)$, $T_2 = (4,1,2)$, $T_3 = (6,2)$.



Optimality of RM and DM

(Section 6.4 of Liu)

Theorem: Neither RM nor DM is optimal.

Proof:

Consider $T_1 = (2,1)$ and $T_2 = (5, 2.5)$.

Total utilization is one, so the system is schedulable.

However, under RM or DM, a deadline will be missed, regardless of how we choose to (statically) prioritize T_1 and T_2.

The details are left as an exercise.

Simply Periodic Systems

Definition: A system of periodic tasks is **simply periodic** if for every pair of tasks T_i and T_k in the system where $p_i < p_k$, p_k is an integer multiple of p_i.

Theorem 6-3: A system **T** of simply periodic, independent, preemptable tasks, whose relative deadlines are at least their periods, is schedulable on one processor according to the RM algorithm if and only if its total utilization is at most one.

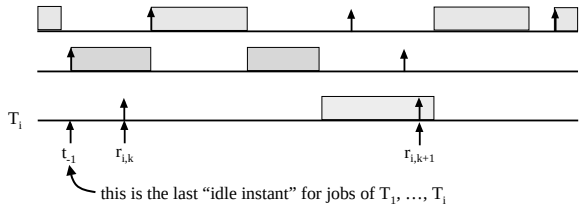
Proof of Theorem 6-3

We wish to show: $U \leq 1 \Rightarrow$ **T** is schedulable.

We prove the contrapositive, i.e., **T** is not schedulable $\Rightarrow U > 1$.

Assume **T** is not schedulable.

Let $J_{i,k}$ be the first job to miss its deadline.



Proof (Continued)

Because $J_{i,k}$ missed its deadline…

the demand placed on the processor in $[t_{-1}, r_{i,k+1})$ by jobs of tasks $T_1, \ldots, T_i$ is greater than the available processor time in $[t_{-1}, r_{i,k+1}]$.

Thus,

$$
\begin{aligned}
& r_{i,k+1} - t_{-1} \\
&= \text{available processor time in } [t_{-1}, r_{i,k+1}] \\
&< \text{demand placed on the processor in } [t_{-1}, r_{i,k+1}) \text{ by jobs of } T_1, \ldots, T_i \\
&= \sum_{j=1}^{i} (\text{the number of jobs of } T_j \text{ released in } [t_{-1}, r_{i,k+1})) \cdot e_j \\
&\leq \sum_{j=1}^{i} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j
\end{aligned}
$$

[Note: Because the system is simply periodic, $\frac{r_{i,k+1} - t_{-1}}{p_j}$ is an integer.]

Proof (Continued)

Thus, we have

$$r_{i,k+1} - t_{-1} < \sum_{j=1}^{i} \frac{r_{i,k+1} - t_{-1}}{p_j} \cdot e_j.$$

Cancelling $r_{i,k+1} - t_{-1}$ yields

$$1 < \sum_{j=1}^{i} \frac{e_j}{p_j},$$

i.e.,

$1 < U_i \leq U$.

This completes the proof.

Optimality Among Fixed-Priority Algs.

Theorem 6-4: A system **T** of independent, preemptable periodic tasks that are in phase and have relative deadlines at most their respective periods can be feasibly scheduled on one processor according to the DM algorithm whenever it can be feasibly scheduled according to any fixed-priority algorithm.

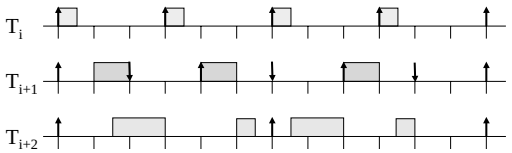
Corollary: The RM algorithm is optimal among all fixed-priority algorithms whenever the relative deadlines of all tasks are proportional to their periods.

Proof of Theorem 6-4

Suppose T_1, …, T_i are prioritized in accordance with DM.

Suppose T_i has a longer relative deadline than T_{i+1}, but T_i has a higher priority than T_{i+1}.

Then, we can interchange T_i and T_{i+1} and adjust the schedule accordingly by swapping "pieces" of T_i with "pieces" of T_{i+1}.

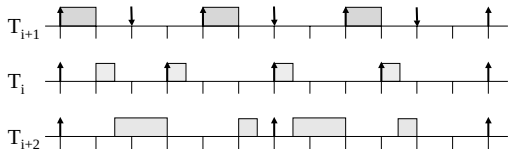


Proof of Theorem 6-4

Suppose T_1, …, T_i are prioritized in accordance with DM.

Suppose T_i has a longer relative deadline than T_{i+1}, but T_i has a higher priority than T_{i+1}.

Then, we can interchange T_i and T_{i+1} and adjust the schedule accordingly by swapping "pieces" of T_i with "pieces" of T_{i+1}.



By induction, we can correct all such situations.

Utilization-based RM Schedulability Test

(Section 6.7 of Liu)

Theorem 6-11: [Liu and Layland] A system of n independent, preemptable periodic tasks with relative deadlines equal to their respective periods can be feasibly scheduled on a processor according to the RM algorithm if its total utilization U is at most

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

Note that this is only a **sufficient** schedulability test.

$U_{RM}(n)$ as a Function of n

n	$U_{RM}(n)$ ← truncated to three digits
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
⋮	⋮
∞	ln 2 ≈ 0.693

Proof Sketch for Theorem 6-11

We will assume that all priorities are distinct, i.e., $p_1 < p_2 < \ldots < p_n$.

Note: The original proof for this theorem by Liu and Layland is incorrect. For a complete, correct proof, see Ed Overton's M.S. thesis on my web page. Overton's thesis also points out where the error is in Liu and Layland's proof.

We will present our proof sketch in two parts:

- First, we consider the special case where $p_n \leq 2p_1$.
- Then, we will remove this restriction.

Special Case: $p_n \leq 2p_1$

Definition: A system is **difficult-to-schedule** if it is schedulable according to the RM algorithm, but it fully utilizes the processor for some interval of time so that any increase in the execution time or decrease in the period of some task will make the system unschedulable.

We seek the **most difficult-to-schedule** system, i.e., the system whose utilization is smallest among all difficult-to-schedule systems.

The proof for the special case $p_n \leq 2p_1$ consists of **four steps**, described next.

Four Steps of the Proof

- **Step 1:** Identify the phases in the most difficult-to-schedule system.
- **Step 2:** Define the periods and execution times for the most difficult-to-schedule system.
- **Step 3:** Show that any difficult-to-schedule system whose parameters are not like in Step 2 has utilization that is at least that of the most difficult-to-schedule system.
- **Step 4:** Compute an expression for $U_{RM}(n)$.

Aside: Critical Instants

Definition: A **critical instant** of a task T_i is a time instant such that:

(1) the job of T_i released at this instant has the maximum response time of all jobs in T_i, if the response time of every job of T_i is at most D_i, the relative deadline of T_i, and

(2) the response time of the job released at this instant is greater than D_i of the response time of some jobs in T_i exceeds D_i.

Informally, a critical instant of T_i represents a worst-case scenario from T_i's standpoint.

Critical Instants in Fixed-Priority Systems

Theorem 6-5: [Liu and Layland] In a fixed-priority system where every job completes before the next job of the same task is released, a critical instant of any task T_i occurs when one of its job $J_{i,c}$ is released at the same time with a job of every higher priority task.

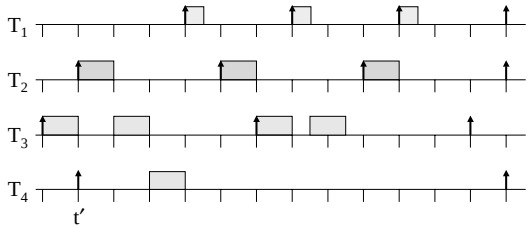
We are not saying that T_1, …, T_i will all necessarily release jobs at the same time, but if this does happen, we are claiming that the time of release will be a critical instant for T_i.

We give a different (probably more hand-waving) proof of Theorem 6-5 than that found in Liu.

Proof of Theorem 6-5

Consider a system such that T_1, …, T_i all release jobs together at some time instant t. Suppose t is not a critical instant for T_i, i.e., T_i has a job released at another time t′ that has a longer response time than its job released at t.

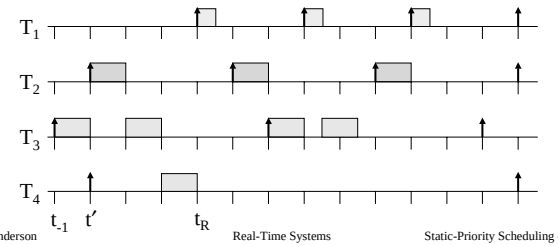
Example:



Proof (Continued)

Let t_{-1} be the latest "idle instant" for $T_1, \ldots, T_i$ at or before t'.
Let J be T_i's job released at t'.
Let t_R denote the time instant when J completes.

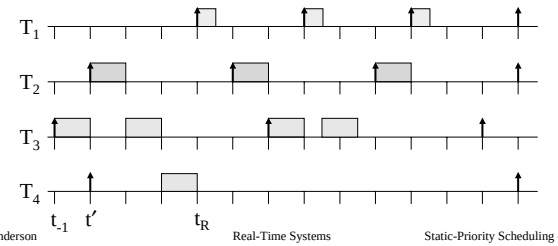
Example:



Proof (Continued)

If we (artificially) redefine J's release time to be t_{-1}, then t_R remains unchanged (but J's response time may increase).

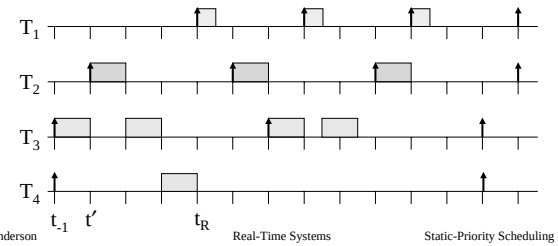
Example:



Proof (Continued)

If we (artificially) redefine J's release time to be t_{-1}, then t_R remains unchanged (but J's response time may increase).

Example:

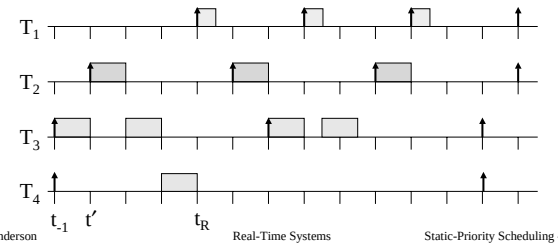


Proof (Continued)

Starting with T_1, let us "left-shift" any task whose first job is released after t_{-1} so that its first job is released at t_{-1}.

With each shift, T_i's response time does not decrease. **Why?**

Example: **Shift over T_1 ...**

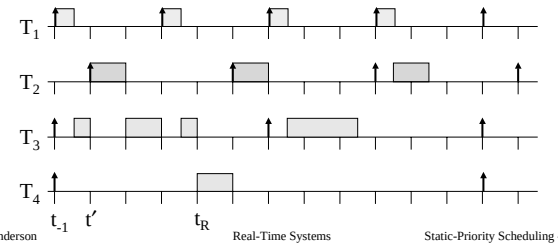


Proof (Continued)

Starting with T_1, let us "left-shift" any task whose first job is released after t_{-1} so that its first job is released at t_{-1}.

With each shift, T_i's response time does not decrease. **Why?**

Example: **Shift over T_2 ...**

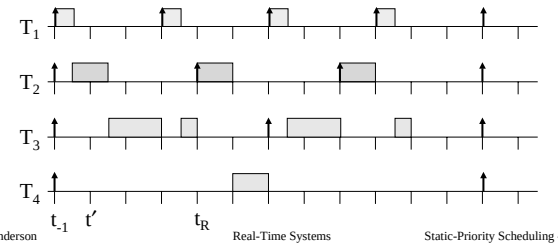


Proof (Continued)

Starting with T_1, let us "left-shift" any task whose first job is released after t_{-1} so that its first job is released at t_{-1}.

With each shift, T_i's response time does not decrease. **Why?**

Example:



Proof (Continued)

We have constructed a portion of the schedule that is identical to that which occurs at time t (when T_1, …, T_i all release jobs together).

Moreover, the response time of T_i's job released at t is at least that of T_i's job released at t′.

This contradicts our assumption that T_i's job released at t′ has a longer response time than T_i's job released at t.

Thus, t is a critical instant.

Step 1: Phases

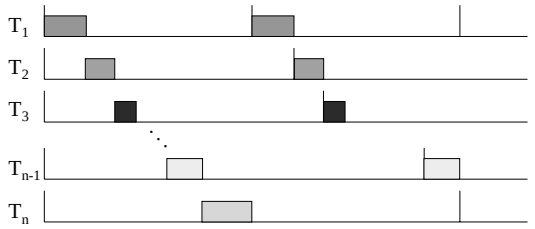
- Back to the proof of Theorem 6-11…
- Recall that Step 1 is to identify the phases in the most difficult-to-schedule system.
- By Theorem 6-5, we can assume that each task in the most difficult-to-schedule system releases its first job at time 0.

Step 2: Periods and Execution Times

- By Theorem 6-5, we can limit attention to the first period of each task.
- We need to make sure that each task's first job completes by the end of its first period.
- We will define the system's parameters so that the tasks keep the processor busy from time 0 until at least p_n, the end of the first period of the lowest-priority task.

Step 2 (Continued)

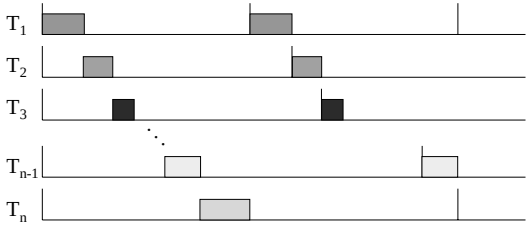
Let us define $e_k = p_{k+1} - p_k$ for k = 1, 2, …, n-1
$e_n = p_n - 2\sum_{k=1,\ldots,n-1} e_k$.



Step 2 (Continued)

Notes:

- This task system is difficult-to-schedule. **(Why?)**
- The processor is fully utilized up to p_n.



Step 3: Showing it's the *Most* D-T-S

We still need to show that the system from Step 2 is the *most* difficult-to-schedule system.

We must show that other difficult-to-schedule systems have equal or higher utilization.

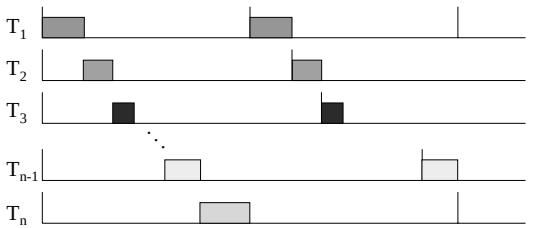
Other difficult-to-schedule systems can be obtained from the one in Step 2 by systematically increasing or decreasing the execution times of some of the tasks.

We show that any small increase or decrease results in a utilization that's at least as big as that of the original task system.

(Convince yourself that this argument generalizes.)

Step 3 (Continued)

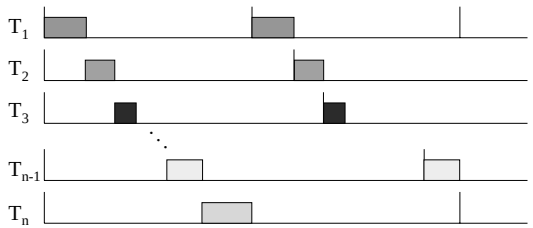
Let's **increase** the execution of some task, say T_1, by ε, i.e.,
$e'_1 = p_2 - p_1 + \varepsilon = e_1 + \varepsilon$.
We can keep the processor busy until p_n by decreasing some T_k's, $k \neq 1$, execution time by ε:
$e'_k = e_k - \varepsilon$.



Step 3 (Continued)

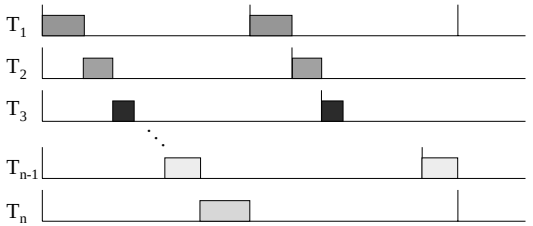
The **difference in utilization** is:

$$U' - U = \frac{e'_1}{p_1} + \frac{e'_k}{p_k} - \frac{e_1}{p_1} - \frac{e_k}{p_k}$$
$$= \frac{\varepsilon}{p_1} - \frac{\varepsilon}{p_k}$$
$$> 0 \quad \text{since } p_1 < p_k$$



Step 3 (Continued)

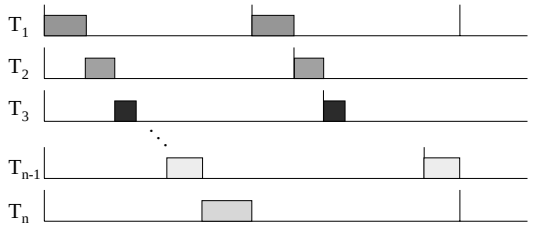
Let's **decrease** the execution of some task, say T_1, by ε, i.e.,
$e''_1 = p_2 - p_1 - \varepsilon$.
We can keep the processor busy until p_n by increasing some T_k's, $k \neq 1$, execution time by 2ε:
$e''_k = e_k + 2\varepsilon$.



Step 3 (Continued)

The **difference in utilization** is: $U'' - U = \frac{2\varepsilon}{p_k} - \frac{\varepsilon}{p_1}$

≥ 0 since $p_k \leq 2p_1$



Step 4: Calculate $U_{RM}(n)$

Let $U(n) = \sum_{k=1}^{n} \frac{e_k}{p_k}$ denote the utilization of the system in Step 2.

Define $q_{k,j} = p_k/p_i$. Then,

$$U(n) = q_{2,1} + q_{3,2} + \cdots + q_{n,(n-1)} + \frac{2}{q_{2,1}q_{3,2}\cdots q_{n,(n-1)}} - n.$$

To find the minimum, we take the partial derivative of U(n) with respect to each adjacent period ratio $q_{k+1,k}$ and set the derivative to zero. This gives us the following n - 1 equations

$$1 - \frac{2}{q_{2,1}q_{3,2}\cdots q_{(k+1),k}^{2}\cdots q_{n,(n-1)}} \quad \text{for all } k = 1, 2, ..., n-1.$$

Solving these equations for $q_{(k+1),k}$, we find that U(n) is at its minimum when all the $n-1$ adjacent period ratios $q_{k+1,k}$ are equal to $2^{1/n}$. Thus,

$U(n) = n(2^{1/n} - 1)$.

Removing the $p_n \leq 2p_1$ Restriction

Definition: The ratio $q_{n,1} = p_n/p_1$ is the **period ratio** of the system.

We have proven Theorem 6-11 only for systems with period ratios of at most 2.

To deal with systems with period ratios larger than 2, we show the following

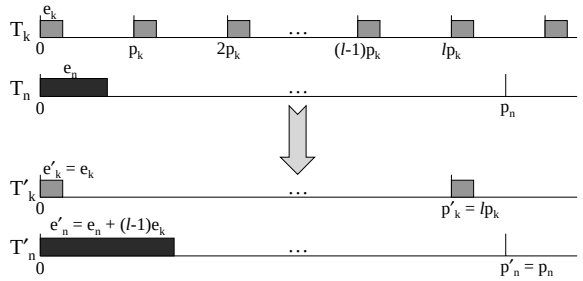
(1) Corresponding to every difficult-to-schedule n-task system **T** whose period ratio is larger than 2 there is a difficult-to-schedule n-task system **T′** whose period ratio is at most 2, and

(2) **T**'s utilization is at least **T′**'s.

Proof of (1)

We show we can transform **T** step-by-step to get **T′**.

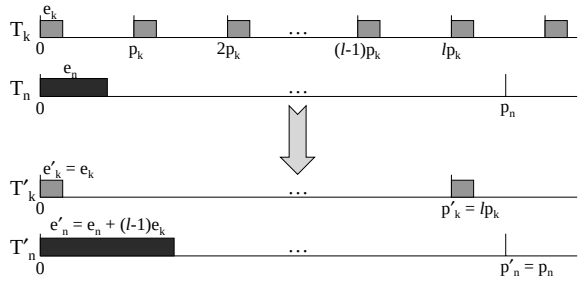
At each step, we find a task T_k whose period is such that $lp_k < p_n \leq (l+1)p_k$, where l is an integer that is at least 2. We modify (only) T_k and T_n as follows.



Proof of (1)

Convince yourself that

- The resulting system is difficult-to-schedule.
- We eventually will get a system with a period ratio of at most 2.



Proof of (2)

It suffices to look at the difference between the utilization of the old and new system when one of the steps in the proof of (1) is applied.

This difference is:

$$\frac{e_k}{p_k} - \frac{e_k}{lp_k} - \frac{(l-1)e_k}{p_n}$$
$$= \left(\frac{1}{lp_k} - \frac{1}{p_n}\right)(l-1)e_k$$
$$> 0 \quad \text{because } lp_k < p_n.$$

This concludes the proof of (2) and (finally!) the proof of Theorem 6-11.

Other Utilization-based Tests

- The book presents several other utilization-based schedulability tests.
  - Some of these tests result in higher schedulable utilizations for certain kinds of task sets.
    - For example, Theorem 6-13 considers systems in which we can partition the tasks into subsets, where in each subset, tasks are simply periodic.
  - Other of these tests allow more detail into the model.
    - For example, Theorem 6-17 considers **multi-frame tasks**, which are tasks where execution costs vary from job to job. (The motivation for this was MPEG video.)
- You will have a better understanding of why people are interested in utilization-based tests later, when we talk about **intractability**.

Time-Demand Analysis

(Section 6.5.2 of Liu)

- **Time-demand analysis** was proposed by Lehoczky, Sha, and Ding.
- 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.

Scheduling Condition

Definition: The time-demand function of the task T_i, denoted $w_i(t)$, is defined as follows.

$$w_i(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \qquad \text{for } 0 < t \le p_i$$

Note: We are still assuming tasks are indexed by priority.

For any fixed-priority **algorithm A** that ensures that each job of every task completes by the time the next job of that task is released…

Theorem: A system **T** of periodic, independent, preemptable tasks is schedulable on one processor by algorithm A if

$$(\forall i :: (\exists t : 0 < t \le p_i :: w_i(t) \le t))$$

holds. This condition is also necessary for synchronous, real-world periodic task systems and also real-world sporadic (= periodic here) task systems.

Sufficiency Proof

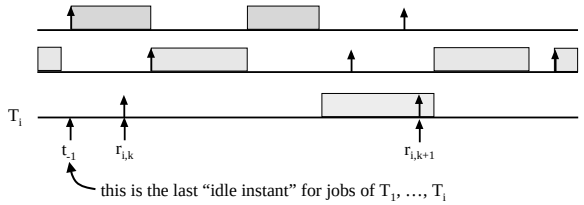
We wish to show: $(\forall i:: (\exists t: 0 < t \le p_i:: w_i(t) \le t)) \Rightarrow$ **T** is schedulable.

We prove the contrapositive, i.e.,

T is not schedulable $\Rightarrow (\exists\, i:: (\forall\, t: 0 < t \le p_i:: w_i(t) > t))$.

Assume **T** is not schedulable.

Let $J_{i,k}$ be the first job to miss its deadline.



Proof (Continued)

Because $J_{i,k}$ missed its deadline…

at all instants t in $(t_{-1}, r_{i,k+1}]$, the demand placed on the processor in $[t_{-1}, t)$ by jobs of tasks $T_1, \ldots, T_i$ is greater than the available processor time in $[t_{-1}, t]$.

Thus, for any t in $(t_{-1}, r_{i,k+1}]$,

$$
\begin{aligned}
& t - t_{-1} \\
&= \text{available processor time in } [t_{-1}, t] \\
&< \text{demand placed on the processor in } [t_{-1}, t) \text{ by jobs of } T_1, \ldots, T_i \\
&= \sum_{j=1}^{i} (\text{the number of jobs of } T_j \text{ released in } [t_{-1}, t)) \cdot e_j \\
&\le \sum_{j=1}^{i} \left\lceil \frac{t - t_{-1}}{p_j} \right\rceil \cdot e_j
\end{aligned}
$$

Proof (Continued)

To recapitulate, we have, for any t in $(t_{-1}, r_{i,k+1}]$,

$$t - t_{-1} < \sum_{j=1}^{i} \left\lceil \frac{t - t_{-1}}{p_j} \right\rceil \cdot e_j$$

Replacing $t - t_{-1}$ by t' in $(0, r_{i,k+1} - t_{-1}]$, we have

$$t' < \sum_{j=1}^{i} \left\lceil \frac{t'}{p_j} \right\rceil \cdot e_j$$

Because $p_i \le r_{i,k+1} - t_{-1}$, we have $(\exists\, i:: (\forall\, t': 0 < t' \le p_i:: w_i(t') > t'))$.

Necessity and Efficiency

- The condition $(\forall i:: (\exists t: 0 < t \le p_i:: w_i(t) \le t))$ is **necessary** for
  - » synchronous, real-world periodic task systems, and
  - » real-world sporadic (= periodic here) task systems.
  - » **Why?**
- For a given i, we don't really have to consider all t in the range $0 < t \le p_i$. **Two ways** to avoid this:
  - » Iterate using "$t^{(k+1)} := w_i(t^{(k)})$", starting with a suitable $t^{(0)}$, and stopping when, for some n, $t^{(n)} \ge w(t^{(n)})$ or $t^{(n)} > p_i$.
  - » Only consider $t = j \cdot p_k$, where $k = 1, 2, \ldots, i$; $j = 1, 2, \ldots, \lfloor \min(p_i, D_i)/p_k \rfloor$.
    - See Liu for an explanation of this.

Fixed-Priority Tasks with Arbitrary Response Times

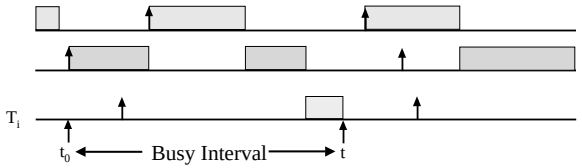
(Section 6.6 of Liu)

- 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 due to Lehoczky 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 FIFO basis.

Busy Intervals

Definition: A **level-π_i busy interval** $(t_0, t]$ begins at an instant t_0 when
(1) all jobs in $\mathbf{T_i}$ released before this instant have completed, and
(2) a job in $\mathbf{T_i}$ is released.
The interval ends at the first instant t after t_0 when all jobs in T_i released since t_0 are complete.

Example:



Busy Intervals (Continued)

- For any t that would qualify as the end of a level-π_i busy interval, a corresponding t_0 exists.
  - » **Why?**
- During a level-π_i busy interval, the processor only executes tasks in $\mathbf{T_i}$ — other tasks can be ignored.
- **Definition:** We say that a level-π_i busy interval is **in phase** if the first job of all tasks that execute in the interval are released at the same time.

It Ain't So Easy …

For systems in which each task's relative deadline is at most its period, we argued that an upper bound on a task's response time could be computed by considering a "critical instant" scenario in which that task releases a job together with all higher-priority tasks.

In other words, we just consider the first job of each task in an in-phase system.

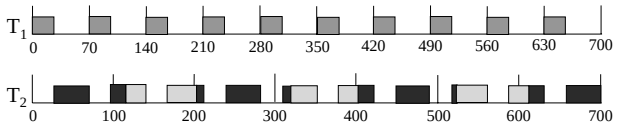
For many years, people just assumed this approach would work if a task's relative deadline could exceed its period.

Lehoczky showed that this "folk wisdom" — that only each task's first job must be considered — is false by means of a counterexample.

Lehoczky's Counterexample

Consider: $T_1 = (70, 26)$, $T_2 = (100, \underline{62})$ [Note: the book has a typo here.]

Here's a schedule:



T_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.

Bottom Line: We have to consider all jobs in an in-phase busy interval.

General TDA Method

Test one task at a time starting with the highest-priority task T_1 in order of decreasing priority. For the purpose of determining whether a task T_i is schedulable, assume that all the tasks are in phase and the first level-π_i busy interval begins at time zero.

While testing whether all the jobs in T_i can meet their deadlines (i.e., whether T_i is schedulable), consider the subset $\mathbf{T_i}$ of tasks with priorities π_i or higher.

(i) If the first job of every task in $\mathbf{T_i}$ completes by the end of the first period of that task, check whether the first job $J_{i,1}$ in T_i meets its deadline. T_i is schedulable if $J_{i,1}$ completes in time. Otherwise, T_i is not schedulable.

(ii) If the first job of some task in $\mathbf{T_i}$ does not complete by the end of the first period of the task, do the following.

(a) Compute the length of the in-phase level-π_i busy interval by solving the equation $t = \sum_{k=1,\ldots,i} \lceil t/p_k \rceil e_k$ iteratively, starting from $t^{(1)} = \sum_{k=1,\ldots,i} e_k$ until $t^{(l+1)} = t^{(l)}$ for some $l \geq 1$. The solution $t^{(l)}$ is the length of the level-π_i busy interval.

(b) Compute the maximum response times of all $\lceil t^{(l)}/p_i \rceil$ jobs of T_i in the in-phase level-π_i busy interval in the manner described below and determine whether they complete in time. T_i is schedulable if all of these jobs complete in time; otherwise T_i is not schedulable.

Computing Response Times

Computing the response time of T_i's first job is almost like before.

The time demand function is defined as follows.

$$w_{i,1}(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \qquad \text{for } 0 < t \le w_{i,1}(t)$$

The only difference from before.

The maximum response time $W_{i,1}$ of $J_{i,1}$ is equal to the smallest t that satisfies the equation $t = w_{i,1}(t)$.

Can do this computation iteratively, as describe before. (Is termination a problem?)

Response Times (Continued)

Lemma 6-6: The maximum response time $W_{i,j}$ of the j-th job of T_i in an in-phase level-π_i busy period is equal to the smallest value of t that satisfies the equation

$$t = w_{i,j}(t + (j-1)p_i) - (j-1)p_i$$

where

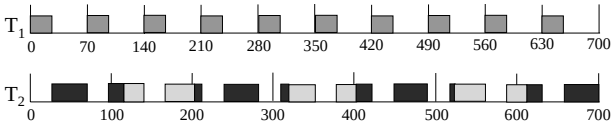
$$w_{i,j}(t) = je_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \qquad \text{for } (j-1)p_i < t \le w_{i,j}(t)$$

The recurrence given in the lemma can be solved iteratively, as described before. (Once again, is termination a problem?)

Example

Let's apply Lemma 6-6 to our previous example:

T_1 = (70, 26), T_2 = (100, 62)



$W_{2,1}$ = minimum t s.t.

$t = w_{2,1}(t)$

$= e_2 + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k$

$= 62 + \left\lceil \frac{t}{70} \right\rceil \cdot 26$

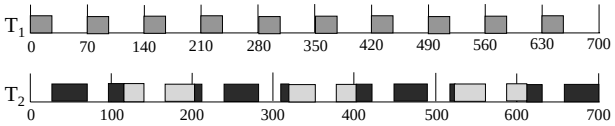
$??114 = 62 + \left\lceil \frac{114}{70} \right\rceil \cdot 26$

$= 62 + 2 \cdot 26$

$= 114$ Yes!

Example (Continued)

T_1 = (70, 26), T_2 = (100, 62)



$W_{2,2}$ = minimum t s.t.

$t = w_{2,2}(t + p_2) - p_2$

$= 2 \cdot e_2 + \sum_{k=1}^{i-1} \left\lceil \frac{t+100}{p_k} \right\rceil \cdot e_k - 100$

$= 124 + \left\lceil \frac{t+100}{70} \right\rceil \cdot 26 - 100$

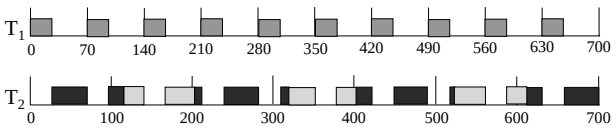
$??102 = 124 + \left\lceil \frac{202}{70} \right\rceil \cdot 26 - 100$

$= 124 + 3 \cdot 26 - 100$

$= 102$ Yes!

Example (Continued)

$T_1 = (70, 26)$, $T_2 = (100, 62)$



$$W_{2,3} = \text{minimum } t \text{ s.t.}$$

$$t = w_{2,3}(t + 2 \cdot p_2) - 2 \cdot p_2$$

$$= 3 \cdot e_2 + \sum_{k=1}^{i-1} \left\lceil \frac{t + 200}{p_k} \right\rceil \cdot e_k - 200$$

$$= 186 + \left\lceil \frac{t + 200}{70} \right\rceil \cdot 26 - 200$$

$$??116 = 186 + \left\lceil \frac{316}{70} \right\rceil \cdot 26 - 200$$

$$= 186 + 5 \cdot 26 - 200$$

$$= 116 \quad \text{Yes!}$$

Correctness of the General Schedulability Test

The general schedulability test hinges upon the assumption that the job with the maximum response occurs within an in-phase busy interval.

We must confirm that this is so.

Aside: In steps (i) and (ii)-(b), the conclusion that "T_i is not schedulable" is stated. In other words, Liu is presenting this as a necessary and sufficient schedulability test. **Why is it OK to conclude that the test is necessary?**

Correctness (Continued)

Correctness follows from several lemmas…

Lemma 6-7: Let t_0 be a time instant at which a job of every task in $\mathbf{T_i}$ is released. All the jobs in $\mathbf{T_i}$ released prior to t_0 have been completed at t_0.

Proof Sketch:

Let t_{-1} be the beginning of the latest busy interval.

Demand created by jobs of tasks in $\mathbf{T_i}$ released in $[t_{-1}, t_0)$ must be fulfilled in $[t_{-1}, t_0]$, or else U_i exceeds 1.

This is similar to the proof of Theorem 6-3.

Note: To reach the stated conclusion about utilization, we must avoid introducing those nasty ceiling operators like in the proof of the original TDA schedulability test. Why is this not a problem here?

Correctness (Continued)

Lemma 6-8: When a system of independent, preemptive periodic tasks is scheduled on a processor according to a fixed-priority algorithm, the time-demand function $w_{i,1}(t)$ of a job in T_i released at the same time with a job in every higher priority task is given by

$$w_{i,1}(t) = e_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{p_k} \right\rceil \cdot e_k \qquad \text{for } 0 < t \le w_{i,1}(t).$$

This should be pretty obvious to you by now …

Correctness (Continued)

Theorem 6-9: The response time $W_{i,j}$ of the j-th job of T_i executed in an in-phase level-π_i busy interval is no less than the response time of the j-th job of T_i executed in any level-π_i busy interval.

This is just the good old critical instant argument …

Correctness (Continued)

Lemma 6-10: The number of jobs in T_i that are executed in an in-phase level-π_i busy interval is never less than the number of jobs in this task that are executed in a level-π_i busy interval of arbitrary phase.

Intuitively, demand on the processor is maximal for a busy interval that is in phase.

Thus, an in-phase busy interval should never be "shorter" than an arbitrary busy interval.

Wrapping Up

- Convince yourself that Lemmas 6-7 through 6-10 imply that we need only look at the jobs that execute within an in-phase busy interval.
- Think carefully about necessity.
  - » For which task models is the schedulability test necessary?
  - » For which is it not necessary?