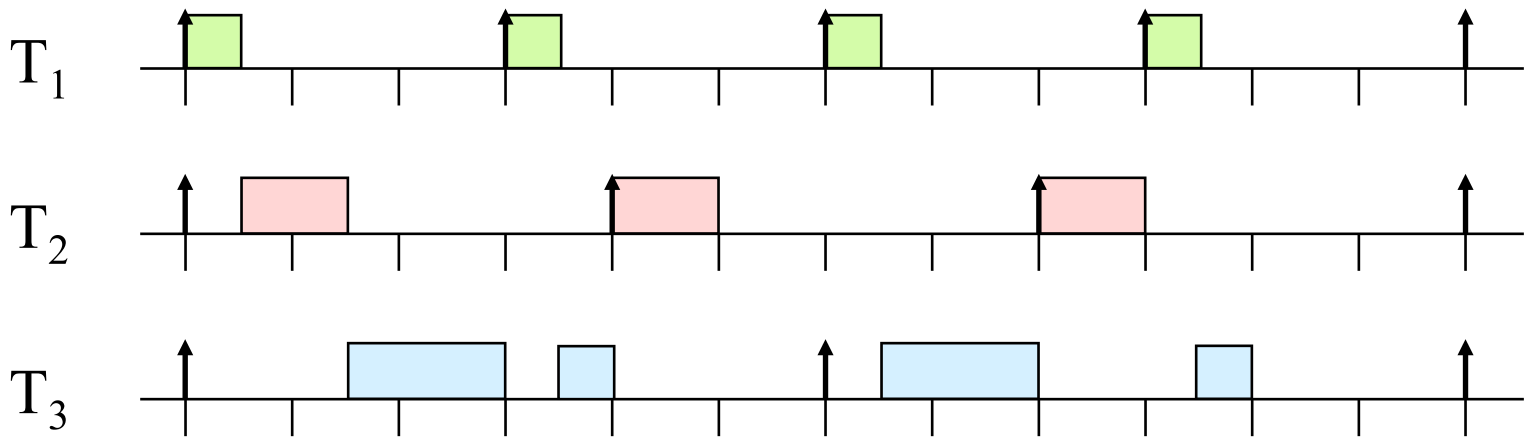
Static-priority scheduling

- As the name suggests…
  - All jobs of a task are assigned the same priority
  - Assume tasks are indexed in decreasing priority order
    - 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:** Typically, it is assumed no two tasks have the same priority
  - So, what task characteristics can we work with?
    - Period and *Relative* deadline

Using period to determine priority…

Rule: Smaller **period** → higher priority

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

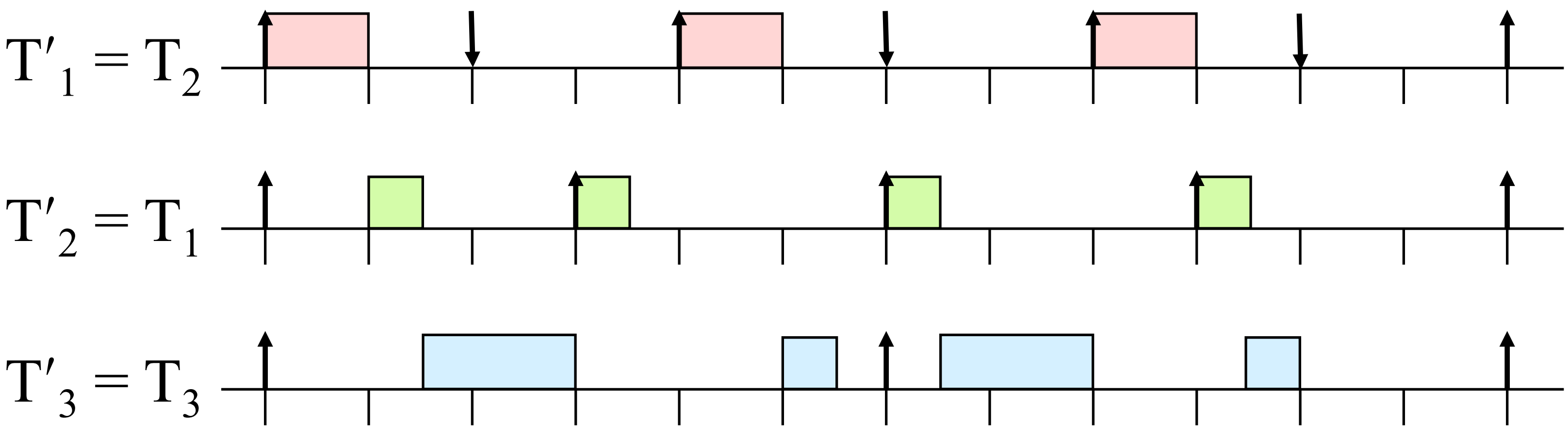


This is called **Rate Monotonic** scheduling

Using relative deadline…

Rule: Smaller **relative deadlines** → higher priority

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



Surprise surprise…this is called **Deadline Monotonic** scheduling

Example

- Derive RM and DM schedules for task set below
  - T1(8, 2, 6), T2(2, 5, 1, 5), T3(10, 2) and T4(7, 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 1

However, under RM or DM, a deadline will be missed, regardless of how priorities are assigned to T_1 and T_2

Schedulability test for RM…

- Build set of *n* tasks that tests the limits of schedulability

- Need to assign
  - Phases
  - Periods
    - Assume all task periods are distinct, i.e., $p_1 < p_2 < \ldots < p_n$
    - Assume relative deadlines equal to periods
  - Execution times

Step 1: Phases of tasks

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

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

(2) response time of the job released at this instant is greater than D_i if response time of even one job of T_i exceeds D_i

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 jobs $J_{i,c}$ is released at the same time with a job of every higher priority task.

So… assign phase of 0 to all tasks

Step 2: Periods and Execution Times

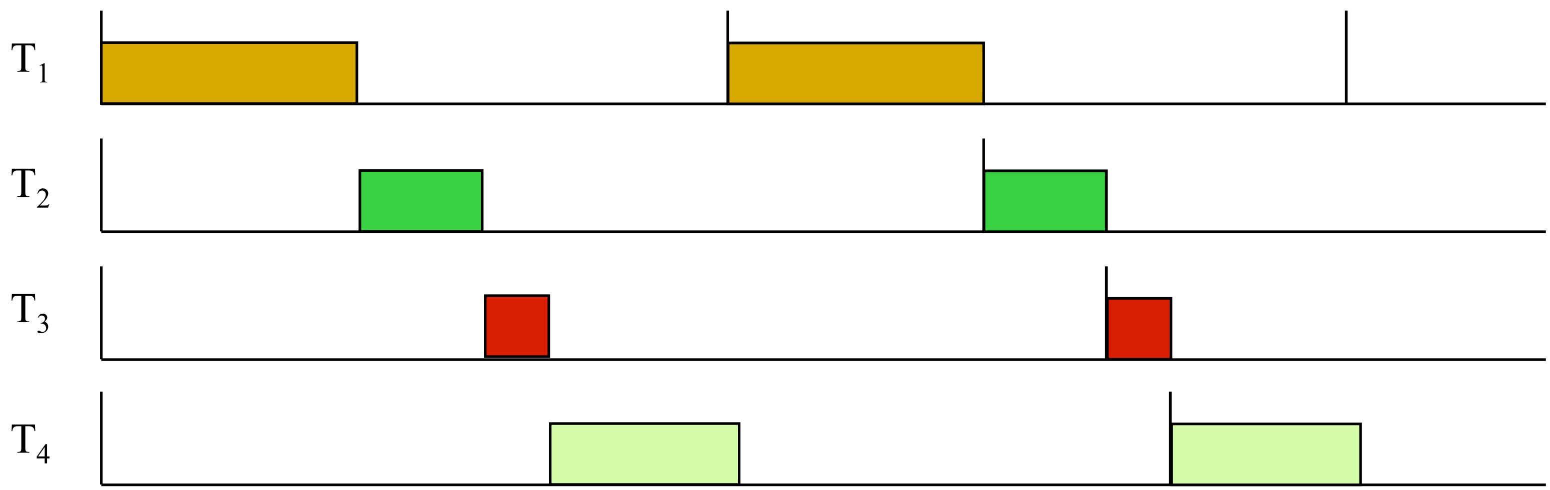
- Limit attention to the first period of each task (Theo 6-5)
- Make sure that each task's first job completes by end of its first period
- Define periods & exec times so that processor is busy from time 0 until at least p_n
- Let's start with the simple case where $p_n \leq 2p_1$

Periods and Execution times

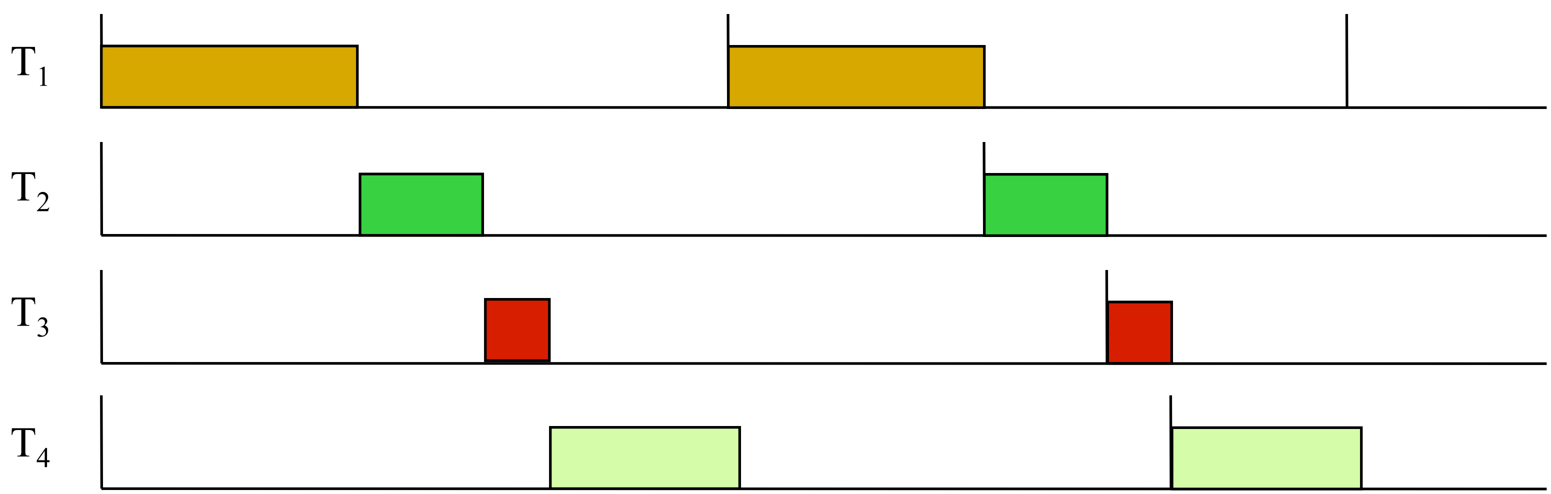
Let us define

$e_k = p_{k+1} - p_k$ for $k = 1, 2, \ldots, n\text{-}1$

$e_n = p_n - 2 \sum_{k = 1, \ldots, n-1} e_k$



- Characteristics of task set we have
  - Schedulable according to RM
  - Fully utilizes processor between **o** & $\mathbf{p_n}$
  - Becomes unschedulable if any task's execution time increases
- Such a system is called a **difficult-to-schedule** system



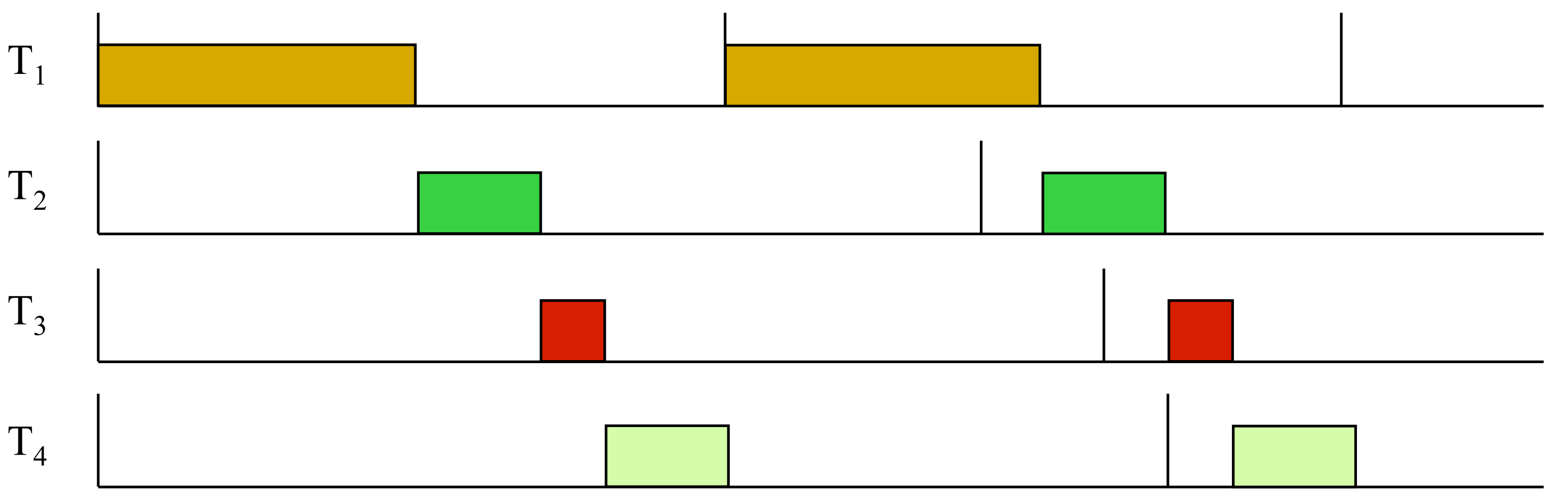
Step 3: Changing the execution times

Increase execution of some task, say T_1, by ε, i.e.,

$$e'_1 = p_2 - p_1 + \varepsilon = e_1 + \varepsilon$$

Can keep processor busy until p_n by decreasing some T_k's ($k \neq 1$), execution time by ε:

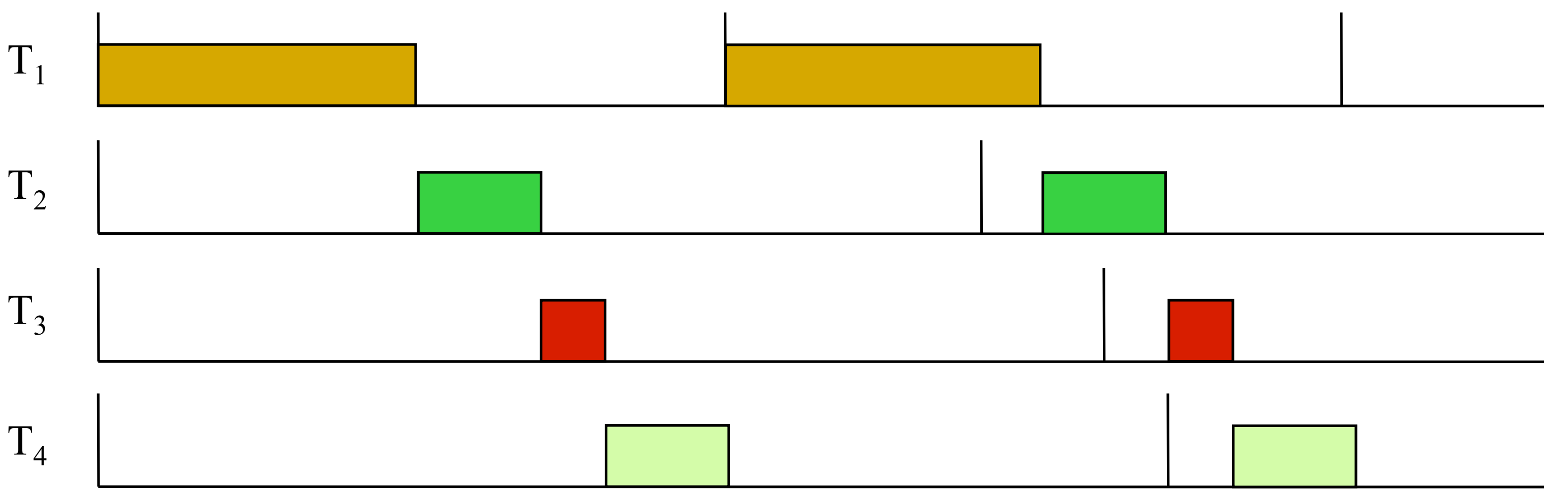
$$e'_k = e_k - \varepsilon$$



How does this affect utilization?

Difference in utilization is:

$$
\begin{aligned}
U' - U &= \frac{e_1'}{p_1} + \frac{e_k'}{p_k} - \frac{e_1}{p_1} - \frac{e_k}{p_k} \\
&= \frac{\varepsilon}{p_1} - \frac{\varepsilon}{p_k} \\
&> 0 \qquad \text{since } p_1 < p_k
\end{aligned}
$$



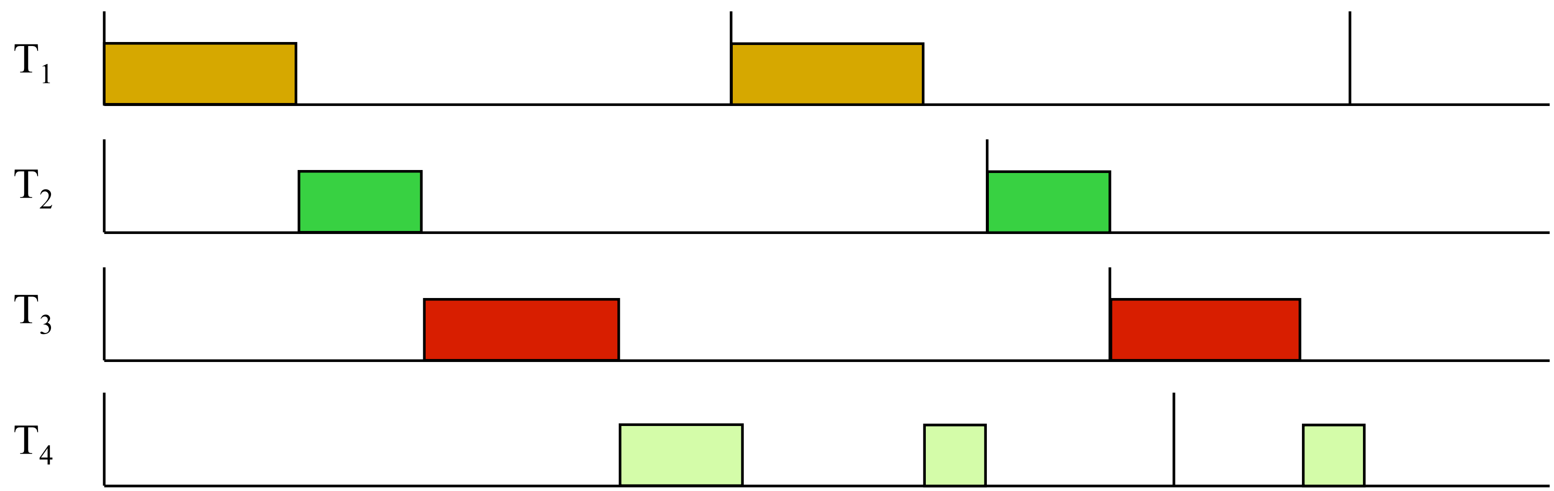
Changing execution times…

Decrease execution of some task, say T_1, by ε, i.e.,

$$e''_1 = p_2 - p_1 - \varepsilon$$

Can keep processor busy until p_n by increasing some T_k's ($k \neq 1$), execution time by 2ε:

$$e''_k = e_k + 2\varepsilon$$

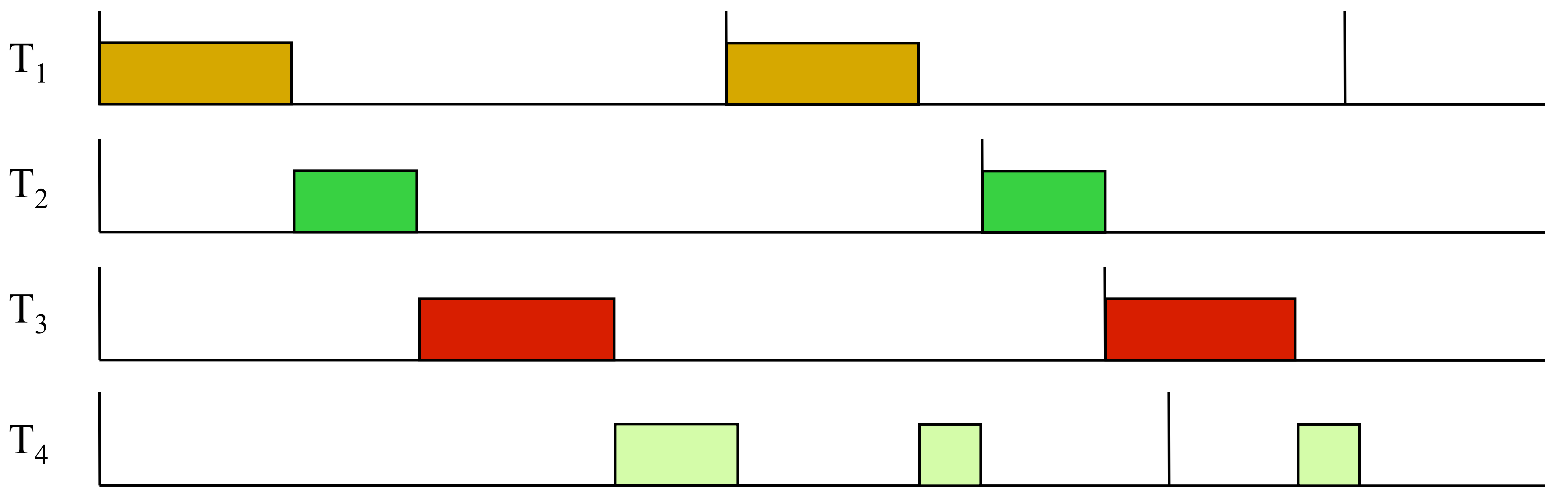


How does this affect utilization?

Difference in utilization is:

$$U'' - U = \frac{2\varepsilon}{p_k} - \frac{\varepsilon}{p_1}$$

$$\geq 0 \qquad \text{since } p_k \leq 2p_1$$



What have we shown so far?

- All systems we've built so far are difficult-to-schedule
- Other difficult-to-schedule systems can be obtained from the one we started with by increasing/decreasing execution times of some tasks
- Any small increase or decrease results in a utilization that's *at least as big* as that of the original task system
- Original one is known as **most-difficult-to-schedule** system
  - Difficult-to-schedule system that has smallest utilization among all difficult-to-schedule systems

Schedulability test for RM

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

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

$$1 - \frac{2}{q_{2,1}q_{3,2}\cdots q^2_{(k+1),k}\cdots q_{n,(n-1)}} = 0 \quad \text{for all } k = 1, 2, \ldots, n-1.$$

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,

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

Schedulability test for RM…

Let's call $U(n)$ $U_{RM}(n)$. **We would like to show:**

$U(\mathbf{T}) \leq U_{RM}(n) \Rightarrow$ "$\mathbf{T}$ is schedulable", where n is the number of tasks in $\mathbf{T}$

As before, this is equivalent to "$\mathbf{T}$ is not schedulable" $\Rightarrow U(\mathbf{T}) > U_{RM}(n)$

- Assume $\mathbf{T}$ is not schedulable
- Let $\mathbf{T}'$ be the same as $\mathbf{T}$, except that all tasks in $\mathbf{T}'$ have a phase of 0
- By "critical instant" theorem, one of the first jobs in $\mathbf{T}'$ misses its deadline
- Suppose $J_{1,1}$, $J_{2,1}$, …, $J_{i-1,1}$ make their deadlines but $J_{i,1}$ misses its deadline
- Recall that $\mathbf{T_i}'$ denotes a subset of $\mathbf{T}'$ with only tasks $T_1 \ldots T_i$
- Let $\mathbf{T_i}''$ consist only of T_1, …, T_i, but reduce T_i's execution cost so that $J_{i,1}$ just barely makes its deadline
- Then $\mathbf{T_i}''$ is a difficult-to-schedule i-task system

Then:

$$U(\mathbf{T}) = U(\mathbf{T}') \geq U(\mathbf{T_i}') > U(\mathbf{T_i}'') \geq U_{RM}(i) \geq U_{RM}(n)$$

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

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

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

n	$U_{RM}(n)$
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
$\vdots$	$\vdots$
∞	$\ln 2 \approx 0.693$

truncated to three digits

Removing the $p_n \le 2p_1$ Restriction

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

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

Proof sketch to deal with systems with period ratios larger than 2

Show that

(1) Every difficult-to-schedule n-task system **T** whose period ratio is larger than 2 there can be transformed to difficult-to-schedule n-task system **T**′ whose period ratio is at most 2

(2)T ’s utilization is at least **T**′’s

Time-Demand Analysis (Section 6.5.2 of Liu)

- **Time-demand analysis** was proposed by Lehoczky, Sha, and Ding.
  - a.k.a. **Response-Time Analysis** (RTA) by Audsley et al.
- Can be applied to any fixed-priority algorithm as long as 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 is necessary *and* sufficient.

Scheduling Condition

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

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

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

For any fixed-priority **algorithm A** with $D_i \leq p_i$ for all i ...

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

$$(\forall i\ (\exists t: 0 < t \leq D_i :: w_i(t) \leq t))$$

holds.

Example

- Calculate response times (i.e., perform Time Demand Analysis) for following tasks under RM priority assignment:

  T1(20, 10), T2(30, 6), T3(40, 8)