Dynamic-priority Scheduling

- The remaining scheduling disciplines we consider are **priority-based**.
  - Each job is assigned a priority, and the highest-priority job executes at any time.
- We begin with **dynamic-priority scheduling**.
  - Under dynamic-priority scheduling, different jobs of a task may be assigned different priorities.
  - Can have the following: job $J_{ik}$ of task $T_i$ has higher priority than job $J_{in}$ of task $T_j$, but job $J_{j,m}$ of $T_j$ has lower priority than job $J_{j,n}$ of $T_j$.
- We will consider static-priority scheduling later.
Outline

- We consider both **earliest-deadline-first (EDF)** and **least-laxity-first (LLF)** (called least-slack-time-first by Liu) scheduling.

**Outline:**

- Optimality of EDF and LLF (Section 4.6 of Liu).
- Utilization-based schedulability test for EDF (Section 6.3 of Liu).

---

Optimality of EDF

**Theorem 4-1:** [Liu and Layland] When preemption is allowed and jobs do not contend for resources, the EDF algorithm can produce a feasible schedule of a set $J$ of independent jobs with arbitrary release times and deadlines on a processor if and only if $J$ has feasible schedules.

**Notes:**

- Applies even if tasks are not periodic.
- If periodic, a task’s relative deadline can be less than its period, equal to its period, or greater than its period.
Proof of Theorem 4-1

We show that any feasible schedule of \( J \) can be systematically transformed into an EDF schedule.

Suppose parts of two jobs \( J_i \) and \( J_k \) are executed out of EDF order:

\[
\begin{array}{cccc}
& J_i & \quad & J_k \\
\uparrow & r_k & \quad & d_k \\
& J_k & \quad & J_i \\
\end{array}
\]

This situation can be corrected by performing a "swap":

\[
\begin{array}{cccc}
& J_k & \quad & J_i \\
\uparrow & r_k & \quad & d_k \\
& J_i & \quad & J_k \\
\end{array}
\]

Proof (Continued)

If we inductively repeat this procedure, we can eliminate all out-of-order violations.

The resulting schedule may still fail to be an EDF schedule because it has idle intervals where some job is ready:

\[
\begin{array}{cccc}
& J_k & \quad & J_i \quad J_k \quad J_i \\
\uparrow & r_k & \quad & d_k \\
& J_i & \quad & J_k \\
\end{array}
\]

Such idle intervals can be eliminated by moving some jobs forward:
**LLF Scheduling**

**Definition:** At any time $t$, the slack (or laxity) of a job with deadline $d$ is equal to $d - t$ minus the time required to complete the remaining portion of the job.

**LLF Scheduling:** The job with the smallest laxity has highest priority at all times.

---

**Optimality of LLF**

**Theorem 4-3:** When preemption is allowed and jobs do not contend for resources, the LLF algorithm can produce a feasible schedule of a set $J$ of independent jobs with arbitrary release times and deadlines on a processor if and only if $J$ has feasible schedules.

The proof is similar to that for EDF and is left as an exercise.

**Question:** Which of EDF and LLF would be preferable in practice?
Preemptive vs. Nonpreemptive EDF

The rest of our discussion of dynamic-priority scheduling will focus on preemptive and non-preemptive EDF. We first show the following:

**Theorem:** Non-preemptive EDF is not optimal.

**Proof:** Consider a system of three jobs \( J_1, J_2, \) and \( J_3 \) such that 
\[
(r_1, e_1, d_1) = (0, 3, 10), (r_2, e_2, d_2) = (2, 6, 14), (r_3, e_3, d_3) = (4, 4, 12).
\]

Here’s a schedule:

<table>
<thead>
<tr>
<th></th>
<th>J₁</th>
<th>J₂</th>
<th>J₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

But under non-preemptive EDF, a deadline is missed!

**Question:** Should we conclude from this result that preemptive EDF is always better than non-preemptive EDF in practice?

**Note:** The EDF optimality proof assumes there is no penalty due to preemption.

**Question:** Are there other practical issues we have ignored?
Utilization-based Schedulability Test for (Preemptive) EDF

Note: Whenever we say “EDF” from now on, we mean preemptive EDF, unless specified otherwise.

Theorem 6-1: [Liu and Layland] A system $T$ 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.

Setting Up the Proof

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 $J_{id}$ be the first job to miss its deadline.

This is the last “idle instant”
Proof (Continued)

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

Thus, $\tau_{k,i+1} - \tau_i$ = available processor time in $[t_i, \tau_{k,i+1}]$
< demand placed on the processor in $[t_i, \tau_{k,i+1}]$ by jobs with deadlines $\leq \tau_{k,i+1}$
= $\sum_{j=1}^{N} (\text{the number of jobs of } T_j \text{ with deadlines } \leq \tau_{k,i+1} \text{ released in } [t_i, \tau_{k,i+1}] ) \cdot c_j$
\leq $\sum_{j=1}^{N} \left[ \frac{\tau_{k,i+1} - \tau_i}{p_j} \right] \cdot c_j$
\leq $\sum_{j=1}^{N} \frac{\tau_{k,i+1} - \tau_i}{p_j} \cdot c_j$

Note: This proof is actually still valid if deadlines are larger than periods.

Proof (Continued)

Thus, we have

$\tau_{k,i+1} - \tau_i < \sum_{j=1}^{N} \frac{\tau_{k,i+1} - \tau_i}{p_j} \cdot c_j$

Cancelling $\tau_{k,i+1} - \tau_i$ yields

$1 < \sum_{j=1}^{N} \frac{c_j}{p_j}$
i.e.,
1 < $U$
This completes the proof.
EDF with Deadlines < Periods

If deadlines are less than periods than \( U \leq 1 \) is no longer a sufficient schedulability condition.

This is easy to see. Consider two tasks such that, for both, \( e_i = 1 \) and \( p_i = 2 \). If both have deadlines at 1.9, then the system is not schedulable, even though \( U = 1 \).

For these kinds of systems, we work with densities instead of utilizations.

**Definition:** The density of task \( T_k \) is defined to be \( \delta_k = \frac{e_k}{\min(D_k, p_k)} \). The density of the system is defined to be \( \Delta = \sum_{k=1}^{N} \delta_k \).

Deadlines < Periods (Continued)

**Theorem 6-2:** A system \( T \) of independent, preemptable, periodic tasks can be feasibly scheduled on one processor if its density is at most one.

The proof is similar to that for Theorem 6-1 and is left as an exercise.

**Note:** This theorem only gives sufficient condition.

We refer to the following as the schedulability condition for EDF:

\[
\sum_{k=1}^{N} \frac{e_k}{\min(D_k, p_k)} \leq 1
\]
Proof of Non-tightness

To see that $\Delta > 1$ doesn’t imply non-schedulability, consider the following example.

**Example:** We have two tasks $T_1 = (2, 0.6, 1)$ and $T_2 = (5, 2.3)$. $\Delta = 0.6/1 + 2.3/5 = 1.06$. Nonetheless, we can schedule this task set under EDF:

```
0 1 2 3 4 5 6 7 8 9 10
T1 ↑ ↓ ↓ ↓ ↓ ↑ ↓ ↓ ↓ ↑
T2 ↑ ↓ ↓ ↓ ↓ ↑ ↓ ↓ ↓ ↑
```

Non-preemptive EDF

*(Jeffay et al.)*

**Theorem:** Let $T = \{T_1, T_2, \ldots, T_n\}$ be a system of independent, periodic tasks with relative deadlines equal to their periods such that the tasks in $T$ are indexed in non-decreasing order by period (i.e., if $i < j$, then $p_i \leq p_j$). $T$ can be scheduled by the non-preemptive EDF algorithm if:

1) $\sum_{i=1}^{n} \frac{e_i}{p_i} \leq 1$
2) $\forall i: 1 \leq i \leq n:: \left( \forall L: p_i < L < p_j: L \geq e_i + \sum_{j=i}^{n-1} \left( \frac{L-1}{p_j} \cdot e_j \right) \right)$

**Note:** This condition is actually necessary and sufficient for “real-world” sporadic tasks.
**Explanation**

The **first condition** is just a constraint on utilization.

In the **second condition**, the term

\[ L \geq e_i + \sum_{j=1}^{i} \left( \frac{L-1}{p_j} \right) e_j \]

gives an upper bound on processor demand in an interval \([t, t+L]\).

**Intuition**: The “worst-case” pattern of job releases occurs when a job of some \(T_i\) begins executing (non-preemptively!) one time unit before some tasks with smaller periods begin releasing some jobs.

These other jobs are **blocked** by the job of \(T_i\).

---

**Second Condition (Continued)**

Here’s an illustration:

For any \(L\) over the range \(p_i < L < p_i\), the total demand on the processor in \([t, t + L]\) due to jobs with deadlines at or before \(t + L\) is:

\[ e_i + \sum_{j=1}^{i} \left( \frac{L-1}{p_j} \right) e_j \]

For the system to be schedulable, this demand must not exceed the length of the interval (which is \(L\)).
Proof of Theorem

Suppose conditions (1) and (2) hold for $T$ but a deadline is missed. Let $t_d$ be the earliest point in time at which a deadline is missed.

There are two cases.

**Case 1:** No job with a deadline after time $t_d$ is scheduled prior to time $t_d$. The analysis is just like with preemptive EDF.

As before, let $t_i$ be the last “idle instant”.

As before, because a deadline is missed at $t_d$, demand over $[t_i, t_d]$ exceeds $t_d - t_i$. In addition, this demand is at most $\sum_{j=1}^{n} \left[ \frac{(t_d - t_i)}{p_j} \right] \cdot e_j$.

Thus, we have $t_d - t_i < \sum_{j=1}^{n} \left[ \frac{(t_d - t_i)}{p_j} \right] \cdot e_j \leq \sum_{j=1}^{n} \left[ \frac{(t_d - t_i)}{p_j} \right] \cdot e_j$.

This implies utilization exceeds one, which contradicts condition (1).

**Case 2:** Some job with a deadline after time $t_d$ is scheduled prior to time $t_d$.

Let $T_i$ be the task with the last job with deadline after $t_d$ that is scheduled prior to $t_d$. Then, we have the following:

![Diagram](image)

Let us bound the processor demand in $[t_i, t_d]$. 

*Proof (Continued)*
Proof (Continued)

Observe the following:

» $p_i > t_d - t_i$.
  • This follows from the fact that the job of task $T_i$ scheduled at time $t_i$ had a deadline after $t_d$.

» No task with index greater than $i$ is scheduled in the interval $[t_i, t_d]$.

» Other than a job of task $T_i$, no job scheduled in $[t_i, t_d]$ could have been released at time $t_i$.

» There is no idle time in the interval $[t_i, t_d]$.

» There is at least one job that is released in $[t_i, t_d]$ with a deadline at or before time $t_d$.

From these facts, we conclude that demand over $[t_i, t_d]$ is less than or equal to

$$c_i + \sum_{j=1}^{i-1} \left( \frac{t_d - (t_i + 1)}{p_j} \right) c_j.$$  

Let $L = t_d - t_i$. Then,

$$L < c_i + \sum_{j=1}^{i-1} \left( \frac{L-1}{p_j} \right) c_j.$$  

This contradicts condition (2).
Notes

- Note that this scheduling condition requires pseudo-polynomial time to evaluate. *(Why?)*
- Using “real-world” terminology, this condition is necessary and sufficient for sporadic and non-concrete periodic task systems. *(Why?)*
  - “Concrete” = fixed release times (though maybe not all 0).
  - For a non-concrete task system to be feasible, it must be schedulable for *any* initial phasing.
- In the rest of the paper, it is shown that the feasibility problem for non-preemptive concrete periodic task systems is NP-hard in the strong sense.
  - Implies that a pseudo-polynomial-time feasibility test is unlikely for such systems. *(We cover this result later when we consider intractability.)*