Resource Access Control in RT Systems

• System Model:
  - Processors(s)
  - m Types of Serially Reusable Resources R1, ..., Rm
  - An execution of a Job Ji requires:
    - A processor for Ji units of time.
    - Some resources for exclusive use.

• Resources:
  - Serially Reusable: Allocated to one job at a time. Once allocated, held by the job until no longer needed.
  - Examples: Semaphores, locks, servers, ....
  - Operations:
    - Lock (Ri)  
    - Unlock (Ri)  
    - Critical section
  - Resources allocated non-preemptively
  - Critical sections properly nested.
Preemption of tasks in their critical sections can cause priority inversion.

**Example:**

```
    T1
    T2
    T3

    lock(s) -> S is locked
    unlock(s)
```

- **Negative effect on schedulability and predictability.**
- **Traditional resource management algorithms fail** (e.g., Banker's algorithm). Decouple resource management decisions from scheduling decisions.
**Predictability: Scheduling Anomalies**

**Example:**

\[ T_1 = (2, 5, 8) \quad T_2 = (4, 7, 22) \quad T_3 = (4, 6, 26) \]

- Missed deadline
- Reduced length of critical section
Al Nok: Disallow Processor Preemption

of Tasks in Critical Section

Define: $\beta$ = Maximum duration of all critical sections.

Analysis identical to analysis with non-prompted
else portions, i.e.

Task $T_i$: schedulable, if

$$\sum_{k=1}^{i} \frac{c_k}{p_k} + \frac{\beta}{p_i} \leq u_x(i)$$

↑ Scheduling Algorithm.

Problem: Critical sections can be rather long.
Priority inheritance can control priority inversion

\[ \pi_1 > \pi_2 > \pi_3 \]

Without priority inheritance

With priority inheritance
**Terminology:**

- A job is **directly blocked** when it requests a resource $R_i$, i.e. executes `lock(R_i)`, but no resource of type $R_i$ is available.

- The scheduler grants the lock request, i.e. allocates the requested resource, to the job, according to the resource allocation rules, as soon as the resources become available.

- $J'$ **directly blocks** $J$ if $J'$ holds some resources when $J$ was requested.

**Priority Inheritance:**

- **Basic strategy for controlling priority inversion:**
  
  Let $\pi$ be the priority of $J$
  
  and $\pi'$ be the priority of $J'$
  
  and $\pi' < \pi$
  
  then the priority of $J'$ to $\pi$ whenever $J'$ to $\pi$ whenever $J'$

- New forms of blocking may be introduced by the resource management policy to control priority inversion and/or prevent deadlock.
BASIC PRIORITY-INHERITANCE PROTOCOL

- Jobs that are not blocked are scheduled according to a priority-driven algorithm preemptively on a processor.

Priorities of tasks are fixed, except for the conditions described below.

A job J requests a resource by executing lock(R)

- If R is available, it is allocated to J, continues executing and releases R by doing unlock(R).

- If R is allocated to J', J' directly blocks J. The request for R is denied.

However: Let t = priority of J when executing lock(R).

\[ t' = \text{priority of } J' \text{ at the same time.} \]

For as long as J' holds R, its priority is max(t, t') and returns to t' when it releases R.

That is: J' inherits the priority of J when J' directly blocks J and J has a higher priority.

- Priority inheritance is transitive.
Example: Priority Inheritance Protocol

\[ P_1 > P_2 > P_3 > P_4 > P_5 \]

Task uses B.

Task uses A.

Task uses A and B.

Problem: If \( T_5 \) tries to lock (B) while it has priority \( P_3 \), we have deadlock!
Properties of Priority Inheritance Protocol

- It does not prevent deadlock.

- A task can be blocked directly by a task with a lower priority at most once, for the duration of the (outmost) critical section.

- Consider a task whose priority is higher than in other tasks

A worst case:

Each of the lower-priority tasks can directly block the task at root once.

A task outside its critical section cannot directly block a higher-priority task.
**Priority Ceiling Protocol**

- **Assumptions:**
  - Priorities of tasks are fixed.
  - Resources required by tasks are known.

- **Therefore:**
  - Priority ceiling of R.
  - We know $T_T = \text{highest priority of all tasks that will require } R$.

- Any task holding $R$ will have priority $T_R$; either its priority is $T_T$, or it inherits $T_T$.

- **Motivation:**
  - Suppose there are resources $A$ and $B$.
  - Both $A$ and $B$ are available. $T_1$ requests $A$.
  - $T_2$ requests $B$ after $A$ is allocated.

  - If $T_2 > T_T$, $T_2$ can never preempt $T_1$.
    - $B$ should be allocated to $T_2$.
  - If $T_2 = T_T$, $T_2$ can preempt $T_1$ (and also request $B$). $B$ should not be allocated to $T_2$ to avoid deadlock.
Priority Ceiling Protocol

- Same as the basic Priority Inheritance Protocol except for the following:
  - When a task \( T \) requests for allocation of a resource \( R \) by executing \texttt{lock}(\( R \)),
  - The request is denied if
    1. \( R \) is already allocated to \( T \) (\( T' \) directly blocks \( T \))
    2. The priority of \( T \) is not higher than all priority ceilings of resources allocated to tasks other than \( T \) at the time \( t \) (these tasks block \( T \)).
  - Otherwise, \( R \) is allocated to \( T \).
  - When a task blocks other tasks, it inherits the highest of their priorities.
EXAMPLE: (FROM LEMOINEY ET AL.
IEEE TC, Sept 1990)

\[ \pi_1 > \pi_2 > \pi_3 \]

\( \pi_X = \pi_2, \quad \pi_Y = \pi_2 = \pi_Z \)

(\( \pi_1 \) LOCK (Z) is denied, since \( \pi_2 = \pi_Y \))
EXAMPLE: PRIORITY CEILING PROTOCOL

(a) FAULT: DIRECTLY BLOCKED BY $T_5$.

(b) FAULT: $P_4 < P_9$

(1) $T_5$ Blocks $T_4$ (to prevent deadlock.)

(2) $T_5$ Blocks $T_3$ (to control priority inversion.)

$P_1 > P_2 > P_3 > P_4 > P_5$

$P_9 = P_2$, $P_8 = P_1$

\[ U(B) \] \[ U(C) \]

\[ U(A) \]

\[ U(A) \]

\[ U(B) \]

\[ U(A) \]

\[ U(A) \]

\[ U(A) \]
Reminders:

- **Blocking**: A higher-priority task waits for a lower-priority task.

- A task \( T_H \) can be blocked by a lower-priority task \( T_L \) in three ways:
  1. **Directly**, i.e.,
     
     \[
     T_H \xrightarrow{\text{Request for}} X \xrightarrow{\text{Allocated to}} T_L
     \]
  2. When \( T_L \) inherits a priority higher than the priority \( P_H \) of \( T_H \):
     
     \[
     T_H \xrightarrow{P_H} X \xrightarrow{\text{Allocated to}} T_L
     \quad \text{\((P > P_H)\)}
     \]
  3. When \( T_H \) requests for a resource, the priority ceiling(s) of resource(s) held by \( T_L \) is equal to or higher than \( P_H \):
     
     \[
     T_H \xrightarrow{\text{Request for}} Y \xrightarrow{P_H \leq P_K} X \xrightarrow{\text{Allocated to}} T_L
     \]
Consider: task $T$ with priority $p_T$ and release time $t$.

**Observation 1:**

$T$ cannot be blocked if at $t$, every resource allocated has a priority ceiling less than $p_T$.

**Obvious:**

- No task with priority lower than $p_T$ holds any resource with priority ceiling $\geq p_T$.

- $T$ will not require any of the resources allocated at time $t$ with priority ceilings $\leq p_T$, and will not be directly blocked waiting for them.

- No lower-priority task can inherit a priority higher than $p_T$ through resources allocated at $t$.

- Requests for resources by $T$ will not be denied because of resource allocations made before $t$.
**Observation 2:**

**Suppose that:**
- There is a task $T_L$ holding a resource $X$.
- $T$ preempts $T_L$, and then
- $T$ is allocated a resource $Y$.

Until $T$ completes, $T_L$ cannot inherit a priority higher or equal to $T$.

**Reason:** ($P_L = \text{priority of } T_L \text{ when it is preempted}$)

- $P_L < P$

- $T$ is allocated a resource $Y$

  $\Rightarrow$ $Y$ is higher than all the priority ceilings of resources held by all lower-priority tasks when $T$ preempts $T_L$.

- $T$ cannot be blocked by $T_L$, from Observation 1.

  $\Rightarrow$ $P_L$ cannot be raised to $P$ or higher through inheritance.
Schedulability analysis with resource access.

Schedulability loss due to blocking:

Reminder: critical sections are properly nested

Duration of a critical section is duration of the outmost critical section

Observation 1:

A low-priority task $T_L$ can block a higher-priority task $T_H$ at most once!

Reason: when $T_L$ is not in critical sections

- $p_L < p_H$
- $T_L$ cannot inherit a higher priority
Observation 2:

A task $T$ can be blocked for at most the duration of one critical section, no matter how many tasks share resources with $T$.

Reason:

- It is not possible for $T$ to be blocked for durations of 2 critical sections of one task.
- It is not possible for $T$ to be blocked by $T_1$ and $T_2$ with priorities $p_1 < p$, $p_2 < p$.

\[ \text{T} \]
\[ \text{U(A)} \quad \text{U(B)} \quad \text{U(A)} \quad \ldots \]

\[ \text{T_1} \]
\[ \text{U(A)} \quad \text{U(B)} \quad \ldots \]

\[ \text{T_2} \]
\[ \text{U(A)} \quad \text{U(B)} \quad \text{U(A)} \quad \ldots \]

↑ NOT POSSIBLE!

$T_2$ is allocated $B \Rightarrow T_2$ is higher than the priority ceiling of $A$, which is $\geq p$.

↑ NOT POSSIBLE!

$T_1$ is allocated $B \Rightarrow T_1$ is not allocated to $T_2$ ($p_1 < p$) AC?

$T_1 \geq p = B$ is not allocated to $T_2$ ($p_1 < p$) AC!
**Observation 3:**

The priority ceiling protocol prevents transitive blocking.

The blocking graph cannot contain a subgraph of the form:

$$
\begin{align*}
T_1 & \rightarrow T_2 \\
(\text{blocked by}) & \rightarrow T_3
\end{align*}
$$

**Reason:**

If such a subgraph were to exist, we must have:

- Tasks’ assigned priorities must satisfy
  $$
  \pi_1 > \pi_2 > \pi_3
  $$

- Two or more resources are involved; the TWF-graph must contain the following subgraph, for two resources $X$ and $Y$.

$$
\begin{align*}
X & \rightarrow T_2 \\
0 & \rightarrow Y \rightarrow T_3
\end{align*}
$$

> It must be that $T_3$ is allocated $Y$, then $T_2$ is allocated $X$.

> From Observation 2: “Until $T_2$ completes, it is not possible for $T_3$ to inherit a priority higher than $\pi_3$.”

- According to the above subgraph, $T_3$ inherits
  $$
  \pi_3 > \pi_2
  $$

$\Rightarrow$ Contradiction!
Observation 4:
Priority ceiling protocol prevents deadlocks.

- Transitive blocking is not possible.
- Therefore, it suffices to show that the blocking graph (or the twin graph) cannot contain cycles of length 2.
  i.e. Subgraphs of the form

\[
\begin{align*}
x \rightarrow y \\
y \rightarrow x
\end{align*}
\]

or

\[
\begin{align*}
x \rightarrow y \\
y \rightarrow x
\end{align*}
\]
Who blocks whom? For how long?

Consider the following resource graph:

- $T_1 = (3, 2, 0.8)$
- $T_2 = (3, 2.1, 0.4)$
- $T_3 = (0, 6, 0.2)$
- $T_4 = (0, 10, 1)$

$T_1 > T_2 > T_3 > T_4$

<table>
<thead>
<tr>
<th>Directly blocked by</th>
<th>Blocked time P.I. by</th>
<th>Blocked due to P.I.</th>
<th>Max. blocking time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>$T_3$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$T_2$ misses deadline at time 2.1 + 3.
Exercise:

\[ T_2 = \{8, 4, [A, 2]\} \]
\[ T_2 \text{ has nested CSS,} \]
\[ \quad \text{nine once for 2 units,} \]
\[ \quad \text{nine once for 4 units.} \]

\[ T_2 = T_3 = \{C, 3\} \]
\[ T_5 = [D, 4, 2, [A, 2], [C, 2.5]] \]
\[ T_4 = [D, 3, [E, 2]] \]

Direct Blocking By

<table>
<thead>
<tr>
<th></th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td>( T_2 )</td>
<td>( T_3 )</td>
<td>( T_4 )</td>
<td>( T_5 )</td>
</tr>
<tr>
<td>( T_2 )</td>
<td>( T_3 )</td>
<td>( T_4 )</td>
<td>( T_5 )</td>
<td></td>
</tr>
<tr>
<td>( T_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Blocking due to P.I. By

<table>
<thead>
<tr>
<th></th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Blocking due to P.C. By

<table>
<thead>
<tr>
<th></th>
<th>( T_2 )</th>
<th>( T_3 )</th>
<th>( T_4 )</th>
<th>( T_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_1 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_2 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_3 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( T_4 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Priority Ceiling of:

\[ A = \]
\[ B = \]
\[ C = \]
\[ D = \]
Worst-case schedulable utilization

A set of $n$ periodic tasks is schedulable by the rate-monotonic algorithm together with the priority-conflicting protocol if

\[
\frac{e_1}{p_1} + \frac{e_2}{p_2} \leq 1
\]

\[
\frac{e_1}{p_1} + \frac{e_2}{p_2} + \frac{e_3}{p_3} \leq 2(\sqrt{2} - 1)
\]

\[
\vdots
\]

\[
\frac{e_1}{p_1} + \frac{e_2}{p_2} + \ldots + \frac{e_i}{p_i} + \frac{b_i}{p_i} \leq i(2^{\frac{1}{2i}} - 1)
\]

\[
\vdots
\]

\[
\frac{e_1}{p_1} + \frac{e_2}{p_2} + \ldots + \frac{e_n}{p_n} \leq n(2^{\frac{1}{2n}} - 1)
\]

- $b_i$ = max. execution time of $T_i = (p_i, e_i)$

- Beliave: the form of this sufficient condition does not give us a complete picture of the worst-case performance.

- Example: maximum number of context switches for request is 4.
**EXAMPLE:**

\[ T_1 = (3, 2, 0.8) \text{ and } [R, 0.8] \]
\[ T_2 = (3, 2.05, 0.8) \]
\[ T_3 = (0, 10, 2) \text{ and } [R, 2] \]

- \( B_2 = 1.0 \)
  \[ C_0 \frac{c_1}{p_1} + \frac{c_2}{p_2} = \frac{0.8 + 1}{2} < 1 \]
  \[ \Rightarrow T_1 \text{ is schedulable.} \]

- \( T_2 \) cannot accept \( T_3 \) in \((0, 2)\) because the priority of \( T_3 \) is \( p_2 \).

\[ B_2 = 1 \implies \frac{c_1}{p_1} + \frac{c_2}{p_2} + \frac{c_3}{p_3} = \frac{0.8 + 1}{2} + \frac{0.2 + 1}{2.05} > 1 \]
  \[ \Rightarrow T_2 \text{ is not schedulable, perhaps.} \]
  \[ \text{(In this case, indeed it is not.)} \]

- \( \frac{c_1}{p_1} + \frac{c_2}{p_2} + \frac{c_3}{p_3} = 0.75 < 3(2.5 - 1) \Rightarrow T_3 \text{ is schedulable.} \)
**EXAMPLE:**

\[ T_1 = (10, 2) \quad B_1 = 3 \]
\[ T_2 = (15, 4) \quad B_2 = 1 \]
\[ T_3 = (25, 3) \quad B_3 = 4 \]
\[ T_4 = (30, 4.5) \]

\[
\frac{e_1}{p_1} + \frac{B_1}{p_1} = \frac{2}{10} + \frac{3}{10} = 0.5 < 1
\]

\[
\frac{e_1}{p_1} \quad \frac{e_2}{p_2} \quad \frac{B_2}{p_2} = \frac{2}{10} + \frac{4}{15} + \frac{3}{15} = 0.533 < 0.832
\]

\[
\frac{e_1}{p_1} \quad \frac{e_2}{p_2} \quad \frac{e_3}{p_2} \quad \frac{B_3}{p_2} = \frac{2}{10} + \frac{3}{15} + \frac{4}{25} = 0.75 < 0.832
\]

\[
\frac{e_1}{p_1} \quad \frac{e_2}{p_2} \quad \frac{e_2}{p_2} \quad \frac{e_4}{p_2} = \frac{2}{10} + \frac{4}{15} + \frac{3}{25} + \frac{4.5}{30} = 0.74 < 0.832
\]

**REMARK:**

*A set of tasks that fails this test may nevertheless be schedulable.*

*Test is very conservative!*
\[ T_1 = (5, 1.5), \; B_1 = 14 \]
\[ T_2 = (6, 2), \; B_2 = 24 \]
\[ T_3 = (7, 3), \; B_3 = 0.5 \]
\[ T_4 = (11, 4) \]
THE MATH:

(1) $1 + 1 < 3.8$ ................................ $T_1$ SCEDULABLE

(2) $2 + 1.5 + 1.4 > 3.8$ ........................ $T_2$ SCEDULABLE

(3) $2 + 1.5 + 1 + 0.5 = 6$ ........................ $T_3$ SCEDULABLE

(4) $2 	imes 1 + 2 	imes 1.5 + 1 + 1.4 < 12$ ........................ $T_4$ SCEDULABLE