Real-Time Systems
Cl.-J. Hamann

Resource Access Control
Problems: Priority Inversion (1)

Assumptions:
Jobs uses resources in a mutually exclusive manner (critical section CS). Preemptive priority-driven scheduling, fixed priorities.

1 processor.

- Busy waiting and priority inversion

```
mutex := 0
```

Declaration
for the following slides
L(R) ↑ U(R) ↓
Problems: Priority Inversion (2)

- Semaphores and priority inversion

M: medium-prioritized job (not using s)
Problems: Timing Anomalies

Reduction of resource usage of $J_3$ by 1.5:
Problems: Deadlocks

• Deadlocks
Assumptions and Notations

1 processor, preemptive priority-driven scheduling, jobs are not self-suspending

- \( R_1, \ldots, R_r \) resources; nonpreemptable, exclusive

- \( L(R_k), U(R_k) \) require/release of \( R_k \); release: LIFO
  \[ \uparrow R_k \quad \downarrow R_k \]

- \( J_1, \ldots, J_n \) jobs
  \( J_h, J_l \): job of high/low priority

- \( p_1, \ldots, p_n \) “assigned” priorities (highest priority: 1);
  w.l.o.g. \( J_i \) ordered according to priorities

- \( p_i(t) \) current priority of \( J_i \)
Assumptions and Notations

- **Jobs conflict with one another**
  require the same resource

- **Jobs contend for a resource**
  one job requests the resource that another job already has

- **Blocked job**
  scheduler cannot grant the requested resource

- **Priority inversion**
  $J_i$ executes while $J_h$ is blocked
Priority Inheritance - Protocol

for preemptive priority-driven scheduling

Sha et al., 1990

- Basic Priority-Inheritance Protocol
  1. Scheduling Rule
     A ready job \( J \) is scheduled according to its current priority \( p(t) \);
     at release time \( t \): \( p(t) := p \).

  2. Allocation Rule
     \( J \) requests \( R \) at time \( t \).
     (a) \( R \) free: \( R \) is allocated to \( J \) until \( J \) releases \( R \).
     (b) \( R \) not free: request is denied, \( J \) becomes blocked.

  3. Priority-Inheritance Rule
     When \( J \) becomes blocked by \( J_i \), so \( J_i \) inherits the current priority of \( J \),
     i.e. \( p_i(t) := p(t) \).
     \( J_i \) executes at this priority until it releases \( R \) at time \( \tilde{t} \).
     Now the priority of \( J_i \) returns to its previous priority:
     \( p_i(\tilde{t}) := p_i(t') \quad t' \): time when \( J_i \) acquire \( R \).
Priority Inheritance - Example

• 2 jobs: no effect!

• 3 jobs:

H ready  H: s.P()  M ready
Properties

- Priority inheritance is transitive.
- No unbounded uncontrolled priority inversion.
- Priority inheritance does not reduce the blocking times as small as possible.
Priority Inheritance – Properties (2)
Priority inheritance does not prevent deadlocks.
Priority Ceilings - Notations

Basic Priority-Ceiling Protocol

Sha/Rajkumar/Lehoczky, 1990

- **Assumptions and Notations**
  - 1 processor, preemptive priority-driven scheduling
    no self-suspension.
  - Assigned priorities $p_i$ are fixed.
    priorities: natural numbers, 1 highest, $\Omega$ lowest priority.
  - The resources required by all jobs are known a priori.

- $P(R)$ **priority ceiling of $R$**
  highest priority of all jobs that require $R$.

- $\hat{P}(t)$ **priority ceiling of the system at time $t$**
  highest priority ceiling of all resources that are in use at time $t$. 
(1) **Scheduling Rule**  
At release time $t^{rel}$ of $J$: $p(t^{rel}) := p$.

(2) **Allocation Rule**  
$J$ requests $R$ at time $t$.  
(a) $R$ held by another job: request denied, $J$ blocked ("on $R$").  
(b) $R$ free:  
   (α) $p(t) > \hat{P}(t)$: $R$ is allocated to $J$.  
   (β) otherwise: $R$ is allocated to $J$ only if $J$ is the job holding the resource(s) $R'$ with $P(R') = \hat{P}(t)$. Otherwise the job becomes blocked.

(3) **Priority-Inheritance Rule**  
When $J$ becomes blocked by $J_i$, $J_i$ inherits $J$'s current priority $p(t)$.  
$J_i$ (preemptive) executes at this priority until it releases every resource whose priority ceiling is at least $p_i(t)$.  
At that time, $J_i$'s priority returns to $p_i(t')$  
($t'$: time when it was granted the resource).
Priority Ceilings - Example
Basic Priority-Ceiling Protocol

\[ S_1 \]
**Priority Ceilings - Properties**

**Basic Priority-Ceiling Protocol**

- **Properties**
  - Difference to priority inheritance: *three* ways to blocking direct:

```
J  -----→  R  -----→  J₁
```

inherence:

```
J  -----→  R  →  J  ___→  J₁
```

ceilings:

```
J  -----→  R  free!          X  -----→  J₁
```

- Deadlocks can never occur.
- There can be no transitive blocking.
A job can be blocked for at most one resource request.

Computation of blocking time - Example:

Priority Ceilings - Example
Basic Priority-Ceiling Protocol

\[ J_1 \quad J_2 \quad J_3 \quad J_4 \]

\[ J_1 \rightarrow R \]
\[ J_2 \rightarrow S \]
\[ J_3 \rightarrow S \]
\[ J_4 \rightarrow R \]

\[ \text{Blocking time} = 0.8 \]

\[ \text{Resource request} = 1 \]

\[ \text{Resource request} = 0.2 \]
Stack-Based Priority-Ceiling Protocol

- **Further Assumptions**
  - Common run-time stack for all jobs (no self-suspending!).
  - Stack space of an active job is on the top of the stack (preemption!).
  - Stack space is freed when the job completes.

- **Protocol**
  
  0. \( \hat{P}(t) = \Omega \), when all \( R \) are free.
     \( \hat{P}(t) \) is updated whenever a resource is allocated or freed.

  1. **Scheduling Rule**
     After \( J \) is released, it is blocked until \( p > \hat{P}(t) \).
     Priority-driven scheduling based on assigned priorities. (!)

  2. **Allocation Rule**
     Whenever a job requests a resource, it is allocated the resource. (!!)

- **Properties**
  - When a job begins to execute, all the resources it will ever need are free.
  - Both protocols result in the same longest blocking time of a job.
  - Deadlocks can never occur.
Preemption-Ceiling Protocol (1)
for Deadline-Driven Systems

- **Properties of EDF**
  - Priorities are static on job level using EDF for periodic tasks.
  - Job $J$ with shorter relative deadline can never be preempted by job $J'$ with larger relative deadline.
    $\rightarrow J_i$ can block $J_h$ only when $J_h$ can preempt $J_i$.

- **Preemption Level (PL) $\pi_j$ of Job $J$**
  - **Requirement**
    It is never possible for $J'$ to preempt $J$ if the PL of $J$ is higher than the PL of $J'$.
  - **Validity Condition**
    If $p > p'$ and $r > r'$ then $\pi > \pi'$. 

Preemption-Ceiling Protocol (2) for Deadline-Driven Systems

- Example

<table>
<thead>
<tr>
<th>Jobs</th>
<th>$p_i$</th>
<th>$r_i$</th>
<th>$e_i$</th>
<th>$\pi_i$</th>
<th>$\pi_i'$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_2$</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_3$</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_4$</td>
<td>4</td>
<td>0</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_5$</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Preemption-Ceiling Protocol (3) for Deadline-Driven Systems

- **EDF-Based Scheduling of Periodic Task Sets**

  A valid preemption-level assignment is according to the relative deadlines $d$ of jobs: the smaller $d$, the higher $\pi$.

  → All jobs in every periodic task in a deadline-driven system have the same preemption level (“fixed preemption-level system”).

- **Preemption Ceilings**

  $\Pi(R)$ preemption ceiling of resource $R$

  $\hat{\Pi}(t)$ preemption ceiling of the system at time $t$

  analogue priority ceilings
Stack-Based, Preemption-Ceiling Protocol

(0) \( \hat{\Pi}(t) = \Omega \), when all \( R \) are free.
\( \hat{\Pi}(t) \) is updated whenever a resource is allocated or freed.

(1) **Scheduling Rule**
After \( J \) is released, it is blocked until
\( \pi_j > \hat{\Pi}(t) \) and \( \pi_j > \pi_{j'} \) (\( j' \): currently executed job).
Priority-driven scheduling based on “assigned" priorities.

(2) **Allocation Rule**
Whenever a job requests a resource, it is allocated the resource.

(3) **Priority-Inheritance Rule**
A job blocking another job from starting inherits the highest priority of all the blocked jobs.
Preemption-Ceiling Protocol (4)
for Deadline-Driven Systems - Example

<table>
<thead>
<tr>
<th>Jobs</th>
<th>$p_i$</th>
<th>$r_i$</th>
<th>$d_i$</th>
<th>$e_i$</th>
<th>$\pi_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_1$</td>
<td>5</td>
<td>13</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_2$</td>
<td>7</td>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_3$</td>
<td>4</td>
<td>17</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_4$</td>
<td>0</td>
<td>24</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$J_5$</td>
<td>3</td>
<td>23</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Exercise 1

5 jobs $J_i$; 
$p_i = i$, $r_i$ release time, $e_i$ execution time 
2 resources $A$, $B$; 
$a_i$ begin of usage (relative to $r_i$) 
$b_i$ duration of usage

<table>
<thead>
<tr>
<th>$i$</th>
<th>$r_i$</th>
<th>$e_i$</th>
<th>$A$: $a_i$</th>
<th>$b_i$</th>
<th>$B$: $a_i$</th>
<th>$b_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>6</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>1.5</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>6</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

To find: schedule and blocking times 

a) without priority inheritance 
b) using priority-inheritance protocol 
c) using priority-ceiling protocol 
d) using stack-based priority-ceiling protocol 
and progress of $\hat{P}(t)$ for c) and d).
Compute the blocking times (disticnted by the reason of blocking) in the case of the given system of jobs!

<table>
<thead>
<tr>
<th></th>
<th>direct</th>
<th>pr. inheritance</th>
<th>pr. ceiling</th>
</tr>
</thead>
<tbody>
<tr>
<td>( J_2 )</td>
<td>( J_3 )</td>
<td>( J_4 )</td>
<td>( J_5 )</td>
</tr>
<tr>
<td>( J_2 )</td>
<td>6</td>
<td>1</td>
<td>( \chi )</td>
</tr>
<tr>
<td>( J_3 )</td>
<td>5</td>
<td>1</td>
<td>( \gamma )</td>
</tr>
<tr>
<td>( J_4 )</td>
<td>3</td>
<td>1</td>
<td>( \zeta )</td>
</tr>
</tbody>
</table>

\( J_1 \)

\( J_2 \)

\( J_3 \)

\( J_4 \)

\( J_5 \)