Problems: Priority Inversion

Assumptions:

- Jobs use resources in a mutually exclusive manner
- Preemptive priority-driven scheduling
- Fixed task priorities
- 1 processor

Busy waiting and priority inversion

Declaration for the following slides

\[ \text{Declaration} \quad \begin{align*} 
L(R) \uparrow & \quad U(R) \downarrow 
\end{align*} \]
Problems: Priority Inversion

Semaphores and priority inversion

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

Reduction of resource usage of \( J_3 \) by 1.5:
Problems: Deadlocks

• exclusive resources
• non-preemptive resources
• sequential acquire
• cyclic wait-condition
Assumptions and Notations

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

- $R_1, \ldots, R_r$ resources; nonpreemptable, exclusive
- $L(R_k), U(R_k)$ acquire/release of $R_k$; release: LIFO
- $J_1, \ldots, J_n$ 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 $J_i$ at time $t$
Assumptions and Notations

**Jobs conflict with one another**
- operate with a common resource

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

**Blocked job**
- scheduler does not 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, $J$ is blocked.

(3) Priority-Inheritance Rule
When $J$ becomes blocked by $J_l$, then $J_l$ inherits the current priority of $J$, i.e. $p_l(t) := p(t)$.
• $J_l$ executes at this priority until it releases $R$ at time $t''$.
• Now the priority of $J_l$ returns to its previous priority:
  $p_l(t'') := p_l(t')$ \quad $t'$: time when $J_l$ acquires $R$. 

Priority Inheritance – Example

- 2 jobs: no effect!
- 3 jobs:

![Diagram showing priority inheritance example]

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

H

M

L
Priority Inheritance – Properties

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
Priority inheritance does not prevent deadlocks.
Priority Ceilings – Notations

Sha/Rajkumar/Lehoczky, 1990

Assumptions and Notations

- 1 processor, preemptive priority-driven scheduling
  no self-suspension
- Assigned priorities $p_i$
  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 $t$
Basic Priority-Ceiling Protocol

(1) **Scheduling Rule**
At release time $t$ of $J$: $p(t) := p$

(2) **Allocation Rule**
$J$ requests $R$ at time $t$.
(a) $R$ held by another job: request denied, $J$ blocks (“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 $J$ blocks

(3) **Priority-Inheritance Rule**
When $J$ becomes blocked by $J_l$, $J_l$ inherits $J$’s current priority $p(t)$
• $J_l$ (preemptively) executes at this priority until it releases every resource whose priority ceiling is at least $p(t)$
• At that time, $J_l$’s priority returns to $p_l(t')$
  ($t'$: time when it was granted the resource)
Basic Priority-Ceiling Protocol – Properties

• Difference to priority inheritance: three ways to blocking
  • direct:
  • inheritance:
  • ceilings:

• Deadlocks can never occur
• There can be no transitive blocking
Basic Priority-Ceiling Protocol – Example

- A job can be blocked for at most one resource request
- Computation of blocking time – Example:

```
J1  0.8
  
J2  1
  0.2

J3  

J4  

R  

S  
```

```
J1  J2  J3  J4
```

(Weights and service times are shown as example values.)
Stack-Based Priority-Ceiling Protocol

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

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

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

(3) Allocation Rule
• Whenever a job requests a resource, it is granted the resource (!)

Properties
• When a job begins execution, all resources it will ever need are free
• Both protocols result in the same longest blocking time of a job
• Deadlocks cannot occur