Shared Resources and Blocking in Real-Time Systems

Giuseppe Lipari
http://feanor.sssup.it/~lipari

Scuola Superiore Sant’Anna – Pisa

October 10, 2008
Outline

1. Priority inversion
2. Priority Inheritance Protocol
3. Priority Ceiling
4. Stack Resource Policy
Blocking and priority inversion

- A blocking condition happens when a high priority task wants to access a resource that is held by a lower priority task.
- Consider the following example, where $p_1 > p_2$.

From time 4 to 7, task $\tau_1$ is blocked by a lower priority task $\tau_2$; this is a priority inversion.

Priority inversion is not avoidable; in fact, $\tau_1$ must wait for $\tau_2$ to leave the critical section.

However, in some cases, the priority inversion could be too large.
Example of priority inversion

- Consider the following example, with $p_1 > p_2 > p_3$.

This time the priority inversion is very large: from 4 to 12.

The problem is that, while $\tau_1$ is blocked, $\tau_2$ arrives and preempt $\tau_3$ before it can leave the critical section.

If there are other medium priority tasks, they could preempt $\tau_3$ as well.

Potentially, the priority inversion could be unbounded!
In the previous example:
In the previous example:
In the previous example:
Example

- In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
Example

- In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
Example

- In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
Example

- In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
In the previous example:

- Task $\tau_3$ inherits the priority of $\tau_1$
- Task $\tau_2$ cannot preempt $\tau_3$ ($p_2 < p_1$)
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.

- $p_1 > p_2 > p_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $\rho_1 > \rho_2 > \rho_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.

- $\rho_1 > \rho_2 > \rho_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.

- $p_1 > p_2 > p_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $\rho_1 > \rho_2 > \rho_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $p_1 > p_2 > p_3$;
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $p_1 > p_2 > p_3$;

At time $t = 7$ task $\tau_3$ inherits the priority of $\tau_2$, which at time 5 had inherited the priority of $\tau_1$. Hence, the priority of $\tau_3$ is $p_1$. 
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.

- $p_1 > p_2 > p_3$;

At time $t = 7$ task $\tau_3$ inherits the priority of $\tau_2$, which at time 5 had inherited the priority of $\tau_1$. Hence, the priority of $\tau_3$ is $p_1$. 
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $p_1 > p_2 > p_3$;

At time $t = 7$ task $\tau_3$ inherits the priority of $\tau_2$, which at time 5 had inherited the priority of $\tau_1$. Hence, the priority of $\tau_3$ is $p_1$. 

![Diagram showing the use of resources and priorities over time]
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.

- $p_1 > p_2 > p_3$;

At time $t = 7$ task $\tau_3$ inherits the priority of $\tau_2$, which at time 5 had inherited the priority of $\tau_1$. Hence, the priority of $\tau_3$ is $p_1$. 
Multiple inheritance

- Task $\tau_1$ uses resource $S_1$; Task $\tau_2$ uses $S_1$ and $S_2$ nested inside $S_1$; Task $\tau_3$ uses only $S_2$.
- $p_1 > p_2 > p_3$;

At time $t = 7$ task $\tau_3$ inherits the priority of $\tau_2$, which at time 5 had inherited the priority of $\tau_1$. Hence, the priority of $\tau_3$ is $p_1$. 
Multiple blocking example

\[ \tau_1 \]

\[ \tau_2 \]

\[ \tau_3 \]

\[ L(S_1) \]

\[ S_1 \]
Multiple blocking example

\[ \tau_1 \]

\[ \tau_2 \]

\[ \tau_3 \]

\[ L(S_1) \]

\[ L(S_2) \]

\[ S_1 \]

\[ S_2 \]
Multiple blocking example

\[ L(S_1) \quad L(S_2) \]

\[ \tau_1 \quad \tau_2 \quad \tau_3 \]
Multiple blocking example
Multiple blocking example

\[ \tau_1 \]

\[ \tau_2 \]

\[ \tau_3 \]

\[ L(S_2) \]

\[ U(S_2) \]

\[ L(S_1) \]

\[ L(S_2) \]

\[ U(S_2) \]

\[ L(S_1) \]

\[ S_1 \]

\[ S_2 \]

\[ S_2 \]

\[ S_2 \]

\[ S_2 \]

\[ L(S_2) \]

\[ U(S_2) \]

\[ L(S_1) \]
Multiple blocking example

\[ L(S_1) \quad U(S_2) \quad L(S_1) \]

\[ L(S_2) \quad U(S_2) \quad S_2 \]

\[ L(S_2) \quad S_2 \quad S_2 \quad U(S_2) \]

\[ L(S_1) \quad S_1 \quad U(S_1) \]
Multiple blocking example
Multiple blocking example
Multiple blocking example

- task $\tau_1$ is blocked twice on two different resources
Example:

$\text{ceil}(S_1) = p_1$ and $\text{ceil}(S_2) = p_1$

- task $\tau_3$ acquires the lock
Example:

\[ \text{ceil}(S_1) = p_1 \text{ and } \text{ceil}(S_2) = p_1 \]

- Task \( \tau_3 \) acquires the lock
- Task \( \tau_2 \) is blocked because \( p_2 < \text{maxceil} = p_1 \)
- Task \( \tau_3 \) inherits \( \tau_2 \)'s priority
Example:

$$\text{ceil}(S_1) = p_1 \text{ and } \text{ceil}(S_2) = p_1$$

- Task $\tau_3$ acquires the lock
- Task $\tau_2$ is blocked because $p_2 < \text{maxceil} = p_1$
- Task $\tau_3$ inherits $\tau_2$’s priority
- Task $\tau_1$ is blocked for the same reason
- Task $\tau_3$ inherits $\tau_1$’s priority
Example:

\[ \text{cei}(S_1) = p_1 \text{ and } \text{cei}(S_2) = p_1 \]

- Task \( \tau_3 \) acquires the lock
- Task \( \tau_2 \) is blocked because \( p_2 < \text{maxcei} = p_1 \)
- Task \( \tau_3 \) inherits \( \tau_2 \)'s priority
- Task \( \tau_1 \) is blocked for the same reason
- Task \( \tau_3 \) inherits \( \tau_1 \)'s priority
- Task \( \tau_3 \) returns to its original priority, since it is not blocking anyone
Example:

\[ \text{ceil}(S_1) = p_1 \text{ and } \text{ceil}(S_2) = p_1 \]

- Task \( \tau_3 \) acquires the lock
- Task \( \tau_2 \) is blocked because \( p_2 < \text{maxceil} = p_1 \)
- Task \( \tau_3 \) inherits \( \tau_2 \)'s priority
- Task \( \tau_1 \) is blocked for the same reason
- Task \( \tau_3 \) inherits \( \tau_1 \)'s priority
- Task \( \tau_3 \) returns to its original priority, since it is not blocking anyone
In the previous example:

- $\tau_1$: $L(S_2)$, $U(S_2)$, $L(S_1)$, $U(S_1)$
- $\tau_2$: $L(S_2)$, $U(S_1)$
- $\tau_3$: $L(S_1)$, $U(S_1)$

The diagram illustrates the blocking events and the intervals for $S_1$ and $S_2$. The horizontal axis represents time, with markers at 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, and 24.
Blocking

In the previous example:

- Blocking time for $\tau_1$: 2
Blocking

In the previous example:

- Blocking time for $\tau_1$: 2
- Blocking time for $\tau_2$: 2
In the previous example:

- Blocking time for $\tau_1$: 2
- Blocking time for $\tau_2$: 2
- No multiple blockings!
Task $\tau_3$ raises the sys ceiling to $p_1$
Example

- Task $\tau_3$ raises the sys ceiling to $p_1$
- Task $\tau_2$ cannot start because $p_2 < \Pi_s = p_1$
Task $\tau_3$ raises the sys ceiling to $p_1$
Task $\tau_2$ cannot start because $p_2 < \Pi_s = p_1$
Task $\tau_1$ cannot start because $p_1 = \Pi_s = p_1$
Example

- Task \( \tau_3 \) raises the sys ceiling to \( p_1 \)
- Task \( \tau_2 \) cannot start because \( p_2 < \Pi_s = p_1 \)
- Task \( \tau_1 \) cannot start because \( p_1 = \Pi_s = p_1 \)
- When task \( \tau_3 \) unlocks, the sys ceiling goes down, and all other tasks can start executing
Task $\tau_3$ raises the sys ceiling to $p_1$

Task $\tau_2$ cannot start because $p_2 < \Pi_s = p_1$

Task $\tau_1$ cannot start because $p_1 = \Pi_s = p_1$

When task $\tau_3$ unlocks, the sys ceiling goes down, and all other tasks can start executing