Real-time Synchronization
(Semaphores, Resources and Blocking)

Priority Inheritance
Priority Ceiling
Slack Resource Policy
The Problem

- Tasks have synchronization constraints
  - Semaphores protect critical sections
- Blocking can cause a higher-priority task to wait on a lower-priority one to unlock a resource
  - Problem: In all previous derivations we assumed that a task can only wait for higher-priority tasks not lower-priority tasks
- Question
  - What is the maximum amount of time a higher-priority task can wait for a lower-priority task?
  - How to account for that time in schedulability analysis?
Mutual Exclusion Constraints

- Tasks that lock/unlock the same semaphore are said to have a mutual exclusion constraint.
Priority Inversion

- Locks and priorities may be at odds. Locking results in priority inversion.
Priority Inversion

- Locks and priorities may be at odds.
  Locking results in priority inversion

High-priority task

Low-priority task

Attempt to lock S results in blocking

Preempt.

Priority Inversion

Lock S
Priority Inversion

- How to account for priority inversion?

Diagram:
- High-priority task
  - Lock S
  - Attempt to lock S results in blocking
  - Preempt.
- Low-priority task
  - Lock S
  - Priority Inversion
  - Unlock S
  - Lock S
  - Unlock S
Unbounded Priority Inversion

Consider the case below: a series of intermediate priority tasks is delaying a higher-priority one.

High-priority task

Intermediate-priority tasks

Low-priority task

Attempt to lock S results in blocking

Unbounded Priority Inversion

Preempt.
Unbounded Priority Inversion

- How to prevent unbounded priority inversion?

High-priority task

Intermediate-priority tasks

Low-priority task

Attempt to lock S results in blocking

Unbounded Priority Inversion

Preempt.
Priority Inheritance Protocol

- Let a task inherit the priority of any higher-priority task it is blocking

**Diagram:**
- High-priority task
  - Attempt to lock S results in blocking
  - Unlock S
- Intermediate-priority tasks
  - Lock S
  - Unlock S
- Low-priority task
  - Lock S
  - Unlock S
Priority Inheritance Protocol

Question: What is the longest time a task can wait for lower-priority tasks?

- Let there be $N$ tasks and $M$ semaphores
- Let the largest critical section of task $i$ be of length $B_i$

Answer: ?
Computing the Maximum Priority Inversion Time

Consider the instant when a high-priority task that arrives.
- What is the most it can wait for lower priority ones?

<table>
<thead>
<tr>
<th>Semaphore Queue</th>
<th>Resource 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semaphore Queue</td>
<td>Resource 2</td>
</tr>
<tr>
<td></td>
<td>Resource M</td>
</tr>
</tbody>
</table>

If I am a task, priority inversion occurs when
(a) Lower priority task holds a resource I need (direct blocking)
(b) Lower priority task inherits a higher priority than me because it holds a resource the higher-priority task needs (push-through blocking)
Maximum Blocking Time

- If all critical sections are equal (of length $B$):
  - Blocking time = $B \min(N, M)$
  (Why?)
- If they are not equal?
Maximum Blocking Time

- If all critical sections are equal (of length $B$):
  - Blocking time = $B \min (N, M)$
  (Why?)
- If they are not equal
  - Find the worst (maximum length) critical section for each resource
  - Add up the top $\min (N, M)$ sections in size
- The total priority inversion time for task $i$ is called $B_i$
Schedulability Test

∀ i, 1 ≤ i ≤ n,

\[
\frac{B_i}{P_i} + \sum_{k=1}^{i} \frac{C_k}{P_k} \leq i(2^{1/i} - 1)
\]
Schedulability Test

\[ \forall i, 1 \leq i \leq n, \]
\[ \frac{B_i}{P_i} + \sum_{k=1}^{i} \frac{C_k}{P_k} \leq i(2^{1/i} - 1) \]

Why do we have to test each task separately? Why not just one utilization-based test like it used to?
Problem: Deadlock

Deadlock occurs if two tasks locked two semaphores in opposite order.

- Lock R1
- Try R1, Block
- Lock R2
- Try R1, Block
- Preemption
- Lock R1
- Try R2, Deadlock
Definition: The priority ceiling of a semaphore is the highest priority of any task that can lock it.

A task that requests a lock $R_k$ is denied if its priority is not higher than the highest priority ceiling of all currently locked semaphores (say it belongs to semaphore $R_h$).

- The task is said to be blocked by the task holding lock $R_h$.

A task inherits the priority of the top higher-priority task it is blocking.
Problem: Deadlock?

Deadlock used to occur if two tasks locked two semaphores in opposite order. Can it still occur in priority ceiling?
Problem: Deadlock?

Deadlock used to occur if two tasks locked two semaphores in opposite order. Can it still occur in priority ceiling?

Lock R2: **Denied because its priority is not higher than ceiling of R1**

- Lock R1
- Lock R2
- Unlock R2
- Unlock R1

Preemption

Inherit higher priority
Maximum Blocking Time

Priority Inheritance Protocol

- Need Red
- Need Blue
- Need Yellow
Maximum Blocking Time

Priority Ceiling Protocol

Need Blue but Priority is lower than Red ceiling

Need Yellow but Priority is lower than Red ceiling

Need Red but Priority is lower than Red ceiling

Done
Schedulability

- A task can be preempted by only one critical section of a lower priority task (that is guarded by a semaphore of equal or higher priority ceiling). Let max length of such section be $B_i$

\[ \forall i, 1 \leq i \leq n, \]
\[ \frac{B_i}{P_i} + \sum_{k=1}^{i} \frac{C_k}{P_k} \leq i(2^{1/i} - 1) \]