Real-Time Operating Systems M

Notice

The course material includes slides downloaded from:

http://codex.cs.yale.edu/avi/os-book/


and

http://retis.sssup.it/~giorgio/rts-MECS.html


which has been edited to suit the needs of this course.

The slides are authorized for personal use only.

Any other use, redistribution, and any for profit sale of the slides (in any form) requires the consent of the copyright owners.
Objectives

- To describe the main problems that may arise in a uniprocessor system when concurrent tasks use shared resources in exclusive mode.
- To present some resource access protocols designed to:
  - avoid these problems
  - bound the maximum blocking time of each task
- To show how such blocking times can be used in schedulability analysis to extend the guarantee test.
For how long will J0 remain blocked on the busy resource?
In general, the blocking time of a task on a busy resource cannot be bounded by the duration of the critical section executed by the lower-priority task.
Priority Inversion: Solutions

- Several methods
  - Fixed priority scheduling: raise the priority of a task when accessing a shared resource
  - EDF: modify a parameter based on the tasks’ relative deadlines

- Fixed priorities:
  - Non-Preemptive Protocol
  - Highest Locker Priority (Immediate Priority Ceiling)
  - Priority Inheritance Protocol
  - Priority Ceiling Protocol

- Dynamic of fixed priorities:
  - Stack Resource Policy
Terminology

- $P_i$: the *nominal* (initial) *priority* of a task
- $p_i \geq P_i$: the *active* priority of a task

- $B_i$: the *maximum blocking time* a task $\tau_i$ can experience

- $z_{i,k}$: a **critical section** of task $\tau_i$ guarded by semaphore $S_k$
  - $Z_{i,k}$: the longest $z_{i,k}$ (for some $\tau_i$ and $S_k$)
  - $\delta_{i,k}$: $Z_{i,k}$'s duration
  - $z_{i,h} < z_{i,k}$: $z_{i,h}$ is *entirely contained* in $z_{i,k}$.

- $\sigma_i$: the set of *semaphores* used by $\tau_i$.
  - $\sigma_{i,j}$: the set of semaphores that can block $\tau_i$, used by a lower-priority $\tau_j$.

- $\gamma_{i,j}$: the set of the **longest critical sections** that can block $\tau_i$, used by a lower-priority $\tau_j$:
  - $\{Z_{i,k} | (P_j < P_i) \text{ and } (S_k \text{ in } \sigma_{i,j})\}$
  - $\gamma_i$: the set of all the longest critical sections that can block $\tau_i$
Assumptions

- $\tau_1, \tau_2, \ldots, \tau_n$ have different priorities and are listed in decreasing order of nominal priority
- No task suspends itself (trap), except on locked semaphores
- Semaphores are mutex
- Critical sections are properly nested (no partially overlapping $z_{i,k}, z_{i,h}$)
Non-Preemptive Protocol (NPP)

“disallow preemption during the execution of any critical section”

- Implementation: the priority of a task entering a critical section is set to highest, and reset to nominal upon leaving the critical section
- NPP solves the priority inversion phenomenon

- Example:

<table>
<thead>
<tr>
<th>(a_i)</th>
<th>(C_i)</th>
<th>(t(R_a))</th>
<th>(z_{i,a})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Non-Preemptive Protocol (NPP)

“disallow preemption during the execution of any critical section”

- Implementation: the priority of a task entering a critical section is set to highest, and reset to nominal upon leaving the critical section
- NPP solves the priority inversion phenomenon

- Example:

<table>
<thead>
<tr>
<th>a_i</th>
<th>C_i</th>
<th>t(R_a)</th>
<th>z_{i,a}</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ_1</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>τ_2</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>τ_3</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
Non-Preemptive Protocol (NPP)

Another example:

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_i)$</th>
<th>$z_{i,a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>8</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
Non-Preemptive Protocol (NPP)

- NPP can create **unnecessary blocking**
- Therefore, NPP is only appropriate when critical sections are short

<table>
<thead>
<tr>
<th></th>
<th>(a_i)</th>
<th>(C_i)</th>
<th>(t(R_a))</th>
<th>(z_{i,a})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
NPP: Blocking Time Computation

- A task $\tau_i$ cannot preempt a lower task $\tau_j$ if $\tau_j$ is inside a critical section.
- Therefore, $\tau_i$ can potentially be blocked by any critical section belonging to any lower task:

$$\gamma_i = \{Z_{j,k} \mid P_j < P_i, \ k = 1, \ldots, m\}$$

- However, only one resource can be locked at any given time. Therefore,

$$B_i = \max_{j,k} \{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

- What are $B_1$, $B_2$, $B_3$ in the previous example?

<table>
<thead>
<tr>
<th></th>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>8</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>
Highest Locker Priority (HLP) Protocol

1) raise the dynamic priority of a task entering a critical section to the highest among those of the tasks sharing that resource ("ceiling" $C(R_k) = p_i(R_k)$)

$$p_i(R_k) = \max_{h} \{ P_h \mid \tau_h \text{ uses } R_k \}$$

2) reset dynamic priority to nominal value upon exiting critical section

Example:

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

- Scheduling?
- How does $p_3$ evolve?
Highest Locker Priority (HLP) Protocol

“1) raise the dynamic priority of a task entering a critical section to the highest among those of the tasks sharing that resource (“ceiling” \( C(R_k) = p_i(R_k) \))

\[
p_i(R_k) = \max_h \{ P_h \mid \tau_h \text{ uses } R_k \}
\]

2) reset dynamic priority to nominal value upon exiting critical section”

- HLP Schedule:

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>( C_i )</th>
<th>( t(R_a) )</th>
<th>( z_{i,a} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>2</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>0</td>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

- Scheduling?
- How does \( p_3 \) evolve?
A task $\tau_i$ can only be blocked by critical sections belonging to lower priority tasks with a semaphore ceiling higher than or equal to $P_i$. Therefore:

$$\gamma_i = \{Z_{j,k} \mid (P_j < P_i) \text{ and } C(R_k) \geq P_i\}$$

Moreover, a task can be blocked at most once. Therefore:

$$B_i = \max_{j,k}\{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$

What are $B_1$, $B_2$, $B_3$ in our example, with NPP and HLP?

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_{a})$</th>
<th>$z_{i,a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>
More Sophisticated Protocols

- **NPP** – idea: upon entering critical section, set priority to highest among all
  - highly pessimistic
    - may block tasks unnecessarily
  - completely transparent and easy to implement

- **HLP** – idea: like NPP, but set priority to highest among all relevant tasks
  - relevant = could be possibly blocked
  - less pessimistic
    - however, may still block tasks unnecessarily
  - less transparent: needs to define ceiling $C(R_k)$

- Improvements:
  - Reduce source of pessimism
    - postpone blocking condition: upon entering $\rightarrow$ upon getting blocked
  - Identify and prevent possible reasons of blocking
  - **Extra features**: address deadlock, multi-unit resources, dynamic priority scheduling, memory utilization
Priority Inheritance Protocol (PIP)

“when a task \( \tau_i \) blocks one or more higher-priority tasks, temporarily assign \( \tau_i \) the highest priority of the blocked tasks”

Protocol in detail:

- Scheduling based on the tasks’ active priorities, and then FCFS;
- When \( \tau_i \) tries to enter \( z_{i,k} \) and \( R_k \) is already held by a lower-priority \( \tau_j \), block \( \tau_i \) and transmit \( \tau_i \)’s priority to \( \tau_j \)
  - \( \tau_j \) inherits the highest priority of the tasks it blocks;
- When \( \tau_j \) exits, unlock \( S_k \), wake up \( \tau_i \), and
  - if no other tasks are blocked by \( \tau_j \), set \( p_j = P_j \);
  - otherwise, set \( p_j \) to the highest priority among the tasks blocked by \( \tau_j \)

Transitivity

- if \( \tau_3 \) blocks \( \tau_2 \), and \( \tau_2 \) blocks \( \tau_1 \), then \( p_3 = P_1 \) (via \( \tau_2 \)).
### Priority Inheritance Protocol (PIP)

<table>
<thead>
<tr>
<th>( a_i )</th>
<th>( C_i )</th>
<th>( t(R_a) )</th>
<th>( z_{i,a} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \tau_1 )</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>( \tau_2 )</td>
<td>4</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>( \tau_3 )</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>
A high-priority task can experience two kinds of blocking:

- **Direct blocking** (from mutual exclusion)
- **Push-through blocking** (from priority inheritance)

What is $\tau_3$’s priority when $\tau_3$ exits the critical section?
Priority Inheritance Protocol (PIP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b)$</th>
<th>$z_{i,b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

- When a task exits a critical section, its active priority is not necessarily restored to its nominal value.
Priority Inheritance Protocol (PIP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b)$</th>
<th>$z_{i,b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>6</td>
<td>1</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

- When a task exits a critical section, its active priority is not necessarily restored to its nominal value.
Priority Inheritance Protocol (PIP)

<table>
<thead>
<tr>
<th></th>
<th>(a_i)</th>
<th>(C_i)</th>
<th>(t(R_a))</th>
<th>(z_{i,a})</th>
<th>(t(R_b))</th>
<th>(z_{i,b})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Example of transitive priority inheritance
**Priority Inheritance Protocol (PIP)**

<table>
<thead>
<tr>
<th></th>
<th>ai</th>
<th>Ci</th>
<th>t(Ra)</th>
<th>zi,a</th>
<th>t(Rb)</th>
<th>zi,b</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ₁</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>τ₂</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>τ₃</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- Example of **transitive priority inheritance**
1. **Lemma.** Push-through blocking can only affect a task $\tau_i$ if a semaphore $S_k$ is accessed both by a task $\tau_h$ with $P_h > P_i$ and a task $\tau_l$ with $P_l < P_i$.

2. **Lemma.** Transitive priority inheritance can occur only in the presence of nested critical sections.

3. **Lemma.** If $l_i$ lower-priority tasks can block $\tau_i$, then $\tau_i$ can be blocked for at most the duration of $l_i$ critical sections (regardless of the number of semaphores used by $\tau_i$).

4. **Lemma.** If $s_i$ semaphores can block $\tau_i$, then $\tau_i$ can be blocked for at most the duration of $s_i$ critical sections (regardless of the number of critical sections used by $\tau_i$).

5. **Theorem.** Under PIP, $\tau_i$ can be blocked for at most the duration of $\alpha_i = \min(l_i, s_i)$ critical sections.
Some Observations about PIP

- Transparency (no need for ceilings)
- Low pessimism (change priority only if problem actually occurs)
- PIP bounds the priority inversion phenomenon
  - Blocking factor $B_i$’s bound = duration of $\alpha_i < \infty$ critical sections

However:

- A precise evaluation of $B_i$ is quite complex, because each critical section of the lower-priority tasks may interfere with $\tau_i$ via direct blocking, push-through blocking, or transitive inheritance.
- A chain of blocking can be formed
  - The blocking duration for a task can still be substantial
- Deadlocks caused by wrong use of semaphores are not prevented by PIP
- Implementation somewhat hard (requires modifying kernel data structures)
## Chained Blocking

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b)$</th>
<th>$z_{i,b}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>-</td>
</tr>
</tbody>
</table>

- How many times can a task be blocked?
### Chained Blocking

**Worst case**: if $\tau_i$ accesses $n$ distinct semaphores that have been locked by $n$ lower-priority tasks, $\tau_i$ will be blocked for the duration of $n$ critical sections.
Deadlock

- That does not depend on PIP
  - but PIP does not do anything to prevent that either…
- Possible solution: total ordering on the semaphore accesses
Exercise 7.4

<table>
<thead>
<tr>
<th>$\delta_{i,R}$</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

- Consider three periodic tasks in decreasing order of priority. The scheduling algorithm is Rate Monotonic. Resources A, B, C are shared using PIP.
- Illustrate the situation produced by PIP+RM in which $\tau_2$ experiences its maximum blocking time
Priority Ceiling Protocol (PCP)

- **Motivation:**
  - To bound the priority inversion phenomenon, **and**
  - To prevent deadlocks and chained blocking

- **Idea:**
  - Add to PIP a **rule for granting a lock request on a free semaphore:**
    - Do not allow a task to enter a critical section if there are **locked semaphores that could block it** (avoid multiple blocking)
    - Thus, once a task enters its first critical section, it can never be blocked by lower-priority tasks until its completion

- **Implementation:**
  - Assign a **priority ceiling** to each semaphore
  - Allow $\tau_i$ to enter $z_{i,k}$ only if $p_i$ is higher than all priority ceilings of the semaphores currently locked by tasks other than $\tau_i$. 
Priority Ceiling Protocol (PCP)

“Grant a task $\tau_j$ access to a critical section only if the task’s active priority is higher than all the nominal priorities of all the tasks that can lock any of the semaphores currently locked by tasks other than $\tau_j$. When a task gets blocked on a semaphore, apply priority inheritance”

- Main points in the protocol:
  - Assign priority ceilings to semaphores (offline)
  - Run the task with highest priority among the ready tasks
  - Guard entrance to critical sections using priority ceiling
  - Assign active priority based on inheritance mechanism

- Transitivity
  - if $\tau_3$ blocks $\tau_2$, and $\tau_2$ blocks $\tau_1$, then $p_3 = p_1$ (via $\tau_2$).
PCP in Detail

1. Assign priority ceilings to semaphores
   - For each $S_k$, $C(S_k) \overset{\text{def}}{=} \max_i \{P_i \mid S_k \in \sigma_i\}$
2. Run the task with highest priority among the ready tasks
   - Let $\tau_i$ be such a task
3. Guard entrance to critical sections using priority ceiling
   - If $\tau_i$ requests access to a critical section guarded by a semaphore:
     - Let $C(S^*)$ be the ceiling of the semaphore $S^*$ with the highest ceiling among all semaphores currently locked by tasks other than $\tau_i$;
     - If $P_i > C(S^*)$ grant access to the critical section
     - Otherwise, if $P_i \leq C(S^*)$, block $\tau_i$;
       - $\tau_i$ is “blocked on $S^*$ by the task that holds the lock on $S^*$”
4. Assign active priority based on inheritance mechanism
   - When $\tau_i$ is blocked on $S^*$, transmit $\tau_i$'s priority to the task $\tau_j$ holding $S^*$
     - $\tau_j$ inherits the highest priority of the tasks it blocks;
   - When $\tau_j$ exits, unlock $S_k$, wake up $\tau_i$, and
     - if no other tasks are blocked by $\tau_j$, set $p_j = P_j$;
     - otherwise, set $p_j$ to the highest priority among the tasks blocked by $\tau_j$
Priority Ceiling Protocol (PCP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>3</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>5</td>
<td>1</td>
</tr>
</tbody>
</table>

- **NPP & HLC Scheduling**
- **PIP Scheduling**

![Diagram of NPP & HLC Scheduling and PIP Scheduling]
Priority Ceiling Protocol (PCP)

<table>
<thead>
<tr>
<th></th>
<th>(a_i)</th>
<th>(C_i)</th>
<th>(t(R_a))</th>
<th>(z_{i,a})</th>
<th>(t(R_b))</th>
<th>(z_{i,b})</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\tau_1)</td>
<td>4</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(\tau_2)</td>
<td>2</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(\tau_3)</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**PIP Scheduling**

- Normal execution
- Critical section
Priority Ceiling Protocol (PCP)

<table>
<thead>
<tr>
<th></th>
<th>a_i</th>
<th>C_i</th>
<th>t(R_a)</th>
<th>z_{i,a}</th>
<th>t(R_b)</th>
<th>z_{i,b}</th>
</tr>
</thead>
<tbody>
<tr>
<td>τ_1</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>τ_2</td>
<td>0</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

- **PIP Scheduling**

- normal execution
- critical section

- τ_1
  - blocked on S_a
  - blocked on S_b

- τ_2
  - t_1, t_2, t_3, t_4, t_5
## Priority Ceiling Protocol (PCP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b)$</th>
<th>$z_{i,b}$</th>
<th>$t(R_c)$</th>
<th>$z_{i,c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Priority Ceiling Protocol (PCP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$t(R_a)$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b)$</th>
<th>$z_{i,b}$</th>
<th>$t(R_c)$</th>
<th>$z_{i,c}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

PCP introduces a third form of blocking: **ceiling blocking**

- It’s necessary, in order to avoid deadlock and chained blocking
PCP: Properties
(Sha, Rajkumar, Lehoczky, 1990)

1. **Lemma.** If a task $\tau_k$ is preempted within a critical section $Z_a$ by a task $\tau_i$ that enters a critical section $Z_b$, then under PCP, $\tau_k$ cannot inherit a priority higher than or equal to that of $\tau_i$ until $\tau_i$ completes.

2. **Lemma.** PCP prevents transitive blocking.

3. **Theorem.** PCP prevents deadlocks.

4. **Theorem.** Under PCP, $\tau_i$ can be blocked for at most the duration of a single critical section.
PCP: Blocking Time Computation

- A task $\tau_i$ can only be blocked by critical sections belonging to lower priority tasks with a semaphore ceiling higher than or equal to $P_i$. Therefore:

  \[ \gamma_i = \{ Z_{j,k} \mid (P_j < P_i) \text{ and } C(R_k) \geq P_i \} \]

- Moreover, a task can be blocked at most once. Therefore:

  \[ B_i = \max_{j,k} \{ \delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i \} \]

- What are $B_1$, $B_2$, $B_3$ in the example below?

<table>
<thead>
<tr>
<th>$\delta_{i,k}$</th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>8</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>6</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
Stack Resource Policy (SRP)

- **Motivation:**
  - To allow multi-unit resources
  - To support dynamic priority scheduling
  - To allow sharing stack-based resources at run-time

- **Idea:**
  - Modify PCP so that a task may be **blocked as it attempts to preempt** (as opposed to: as it makes its first resource request).
SRP: Definitions (Tasks)

- Each $\tau_i$ is characterized by a **priority** $p_i$ and a **preemption level** $\pi_i$
- (Fixed or dynamic) **priorities** $p_i$ represent a task’s importance (urgency)
  - E.g., RM or EDF could be used.
- **Preemption level** $\pi_i$ is a static parameter associated with all instances of each given task, used to inhibit preemption.
  - Idea: $\tau_a$ can preempt $\tau_b$ only if $\pi_a > \pi_b$.
  - General definition: “if $\tau_a$ arrives after $\tau_b$ and $p_a > p_b$, then $\pi_a > \pi_b$”
  - With **EDF**: $\pi_i > \pi_j \iff D_i < D_j$
  - Example:
    - $D_1=10$, $D_2=5$
    - $\pi_1 = 1 < \pi_2 = 2$

(b): $p_1 > p_2$, but $\tau_2$ cannot be preempted (also, there is no need to)
SRP: Definitions (Resources)

- **Units:** $N_k, n_k$
  - Each $R_k$, guarded by $S_k$, is allowed $N_k$ units
  - $n_k$ = number of currently available units for $R_k$
    - Example: if $n_k=0$, a task requiring 3 units of $R_k$ is blocked until $n_k \geq 3$

- **Requirements:** $\text{wait}(S_k, r), \mu_i(R_k)$
  - $\text{wait}(S_k, r)$ When entering $z_{i,k}$, $\tau_i$ specifies how many units it needs
  - $\text{signal}(S_k)$ releases all the $r$ units
  - $\mu_i(R_k)$ is $\tau_i$’s maximum demand with respect to $R_k$ (code analysis)

- **Ceilings:** $C_{Rk}(n_k), \Pi_s$
  - Each $S_k$ is (dynamically) assigned a ceiling $C_{Rk}(n_k)$
  - $C_{Rk}(n_k)$ is the highest preemption level of tasks that could be blocked on $R_k$ if issuing their maximum request when there are only $n_k$ units left
  - $\Pi_s$ is the system ceiling

\[
C_{Rk}(n_k) = \max\{\pi_i \mid \mu_i(R_k) > n_k\} \quad \Pi_s = \max_k\{C_{Rk}\}
\]
Consider three periodic tasks sharing three multi-unit resources.

Compute \( \pi_1 \), \( \pi_2 \), \( \pi_3 \) under EDF, and the ceiling table for the SRP.
Stack Resource Policy (SRP)

“Allow a task to preempt only if its priority is highest among those of all ready tasks, and its preemption level is higher than the system ceiling”

(SRP Preemption Test)

- When a task $\tau_i$ needs a resource that is not available, $\tau_i$ is blocked as it attempts to preempt (rather than later);
- To avoid multiple priority inversions, a task $\tau_i$ is not allowed to start until all currently available resources suffice to meet the maximum need of every task that can preempt $\tau_i$. 
Some Observations about SRP

- Under SRP, once a task starts executing, it will never be blocked for resource contention (Baker’s Theorem).

- The SRP Preemption Test is performed *before* a task starts executing, but resources are actually allocated *only upon request*.

- The SRP Preemption Test can block a task $\tau$ even if $\tau$ does not require any resource.

- The SRP Preemption Test has the effect of imposing priority inheritance without modifying the priority of the task.

- Simple to implement: precompute resource ceilings, use stack for system ceiling, no need for semaphore queues because tasks don’t get blocked.
# Stack Resource Policy (SRP)

<table>
<thead>
<tr>
<th></th>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$t(R_a), \mu_a$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b), \mu_b$</th>
<th>$z_{i,b}$</th>
<th>$t(R_c), \mu_c$</th>
<th>$z_{i,c}$</th>
<th>$N_a$</th>
<th>$N_b$</th>
<th>$N_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>4</td>
<td>7</td>
<td>19</td>
<td>2,1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>1,1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>7</td>
<td>22</td>
<td>6,2</td>
<td>1</td>
<td>2,1</td>
<td>2</td>
<td>1,3</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>25</td>
<td>3,3</td>
<td>2</td>
<td>1,1</td>
<td>5</td>
<td>7,1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Stack Resource Policy (SRP)

<table>
<thead>
<tr>
<th>$a_i$</th>
<th>$C_i$</th>
<th>$D_i$</th>
<th>$t(R_a), \mu_a$</th>
<th>$z_{i,a}$</th>
<th>$t(R_b), \mu_b$</th>
<th>$z_{i,b}$</th>
<th>$t(R_c), \mu_c$</th>
<th>$z_{i,c}$</th>
<th>$N_a$</th>
<th>$N_b$</th>
<th>$N_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>4</td>
<td>7</td>
<td>19</td>
<td>2,1</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>1,1</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>2</td>
<td>7</td>
<td>22</td>
<td>6,2</td>
<td>1</td>
<td>2,1</td>
<td>2</td>
<td>1,3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>0</td>
<td>10</td>
<td>25</td>
<td>3,3</td>
<td>2</td>
<td>1,1</td>
<td>5</td>
<td>7,1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of Stack Resource Policy (SRP)](image_url)
SRP: Properties (Baker, 1991)

1. **Lemma.** If a task $\tau$’s preemption level is greater than the current ceiling of a resource R, then there are sufficient units of R available to meet the maximum requirements of $\tau$ and of every task that can preempt $\tau$.

2. **Theorem.** If no task $\tau$ is permitted to start until $\pi(\tau) > \Pi_s$, then **no task can be blocked after it starts**.
   - This implies that no interpenetration can occur in the stack
   - Therefore, the stack can be shared

3. **Theorem.** The SRP prevents deadlocks.

4. **Theorem.** Under SRP, a task can be **blocked for at most the duration of a single critical section**.
Stack Sharing

- All tasks can share a single stack space.
- When a task is preempted, it maintains its stack and the new task’s stack is allocated immediately above.
- Very useful if many tasks.
SRP: Blocking Time Computation

- Same maximum blocking time as with the PCP.
- Consider worst case for ceiling: $n_k=0$ (no resources available)

$$\gamma_i = \{Z_{j,k} \mid (\pi_j < \pi_i) \text{ and } C_{S_k}(0) \geq \pi_i\}$$

Moreover, a task can be blocked at most once. Therefore:

$$B_i = \max_{j,k} \{\delta_{j,k} - 1 \mid Z_{j,k} \in \gamma_i\}$$
Schedulability Analysis
General Remarks

- All schedulability tests seen so far for independent tasks can be extended to include blocking terms.

- The extended tests guarantee one task $\tau_i$ at a time, by inflating its computation time $C_i$ by the blocking factor $B_i$.

- Tests that were necessary and sufficient under preemptive scheduling are now simply sufficient.
Extended Schedulability Tests (Bounds)

- A task set with $D_i = T_i$ for all tasks is **RM-schedulable** if:
  \[ \forall i = 1, \ldots, n \sum_{h : P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq i(2^{1/i} - 1) \]  
  (Liu-Layland)
  or if:
  \[ \forall i = 1, \ldots, n \prod_{h : P_h > P_i} \left( \frac{C_h}{T_h} + 1 \right) \left( \frac{C_i + B_i}{T_i} + 1 \right) \leq 2 \]
  (Hyperbolic Bound)

- A task set with $D_i = T_i$ for all tasks is **EDF-schedulable** if:
  \[ \forall i = 1, \ldots, n \sum_{h : P_h > P_i} \frac{C_h}{T_h} + \frac{C_i + B_i}{T_i} \leq 1 \]
Extended Analysis Techniques

- Under blocking conditions, the response time of a generic task $\tau_i$ with a fixed priority can be computed by the following recurrent relation:

$$\begin{align*}
R_i^{(0)} &= C_i + B_i \\
R_i^{(s)} &= C_i + B_i + \sum_{h: P_h > P_i} \left[ \frac{R_i^{(s-1)}}{T_h} \right] C_h
\end{align*}$$

- There exist similar extensions for other types analysis (e.g., Processor Demand Criterion).
## Synopsis: Resource Access Protocols

<table>
<thead>
<tr>
<th>priority</th>
<th>num. of blockings</th>
<th>pessimism</th>
<th>blocking instant</th>
<th>transparency</th>
<th>deadlock prevention</th>
<th>implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPP</td>
<td>any</td>
<td>1</td>
<td>high</td>
<td>arrival</td>
<td>YES</td>
<td>easy</td>
</tr>
<tr>
<td>HLP</td>
<td>fixed</td>
<td>1</td>
<td>medium</td>
<td>arrival</td>
<td>NO</td>
<td>easy</td>
</tr>
<tr>
<td>PIP</td>
<td>fixed</td>
<td>$\alpha_i$</td>
<td>low</td>
<td>access</td>
<td>YES</td>
<td>hard</td>
</tr>
<tr>
<td>PCP</td>
<td>fixed</td>
<td>1</td>
<td>medium</td>
<td>access</td>
<td>NO</td>
<td>medium</td>
</tr>
<tr>
<td>SRP</td>
<td>any</td>
<td>1</td>
<td>medium</td>
<td>arrival</td>
<td>NO</td>
<td>easy</td>
</tr>
</tbody>
</table>

- Blocking on arrival (SRP) as opposed to on access (PCP):
  - (Slightly) reduces concurrency + may cause unnecessary blocking 😊
  - Saves unnecessary context switches + reduces runtime overhead 😊
  - Simplifies implementation of protocol 😊
  - Allows sharing of runtime stack resources 😊
Thank You
Exercise

Consider a set of periodic tasks, simultaneously activated at time $t = 0$. There are no resource constraints.

1. What is the task set’s processor utilization?
2. Is the task set DM-schedulable?

<table>
<thead>
<tr>
<th>Task</th>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$D_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>1</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>2</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>$\tau_4$</td>
<td>1</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>
Consider the task set in the table. There are no resource constraints.

1. Is the task set DM-schedulable?
2. Is the task set EDF-schedulable?
Exercise 7.1

Consider the task set in the table. There are resource constraints that may cause blocking as indicated in the table.

1. Verify the RM-schedulability

<table>
<thead>
<tr>
<th>$C_i$</th>
<th>$T_i$</th>
<th>$B_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>3</td>
<td>15</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>
Consider three periodic tasks in decreasing order of priority. Resources A, B, C are shared using PCP.

1. Compute $B_i$ for all tasks.
Exercise 7.8

Consider three periodic tasks sharing three multi-unit resources with the SRP. Scheduling is done using EDF.

1. Compute $\pi_1$, $\pi_2$, $\pi_3$, and the ceiling table.

<table>
<thead>
<tr>
<th></th>
<th>$D_i$</th>
<th>$\mu_a$</th>
<th>$\mu_b$</th>
<th>$\mu_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td>10</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_3$</td>
<td>20</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$N_a$</th>
<th>$N_b$</th>
<th>$N_c$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\tau_1$</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>$\tau_2$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\tau_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>