Scheduling Algorithm and Analysis

RT Synchronization Protocol
(Module 33)

Yann-Hang Lee
Arizona State University
yhlee@asu.edu
(480) 727-7507

Summer 2014
Nonpreemption Protocol

\( \tau_2: \{ \ldots P(S1) \ldots V(S1) \ldots \} \)

\( \tau_4: \{ \ldots P(S1) \ldots V(S1) \ldots \} \)
Advantages and Disadvantages

- **Advantages:**
  - Simplicity
  - Use with fixed-priority and dynamic-priority systems
  - No priori knowledge about resource requirement by each task
  - Good when all critical sections are short

- **Disadvantages:**
  - Every task can be blocked by every lower priority task, even when there is no resource sharing between the tasks.
  - Blocking time: $\max(cs_i)$
Basic Inheritance Protocol (BIP)

\[ \tau_1(H) \]

\[ \tau_2 \]

\[ \tau_3 \]

\[ \tau_4(L) \]

\[ \tau_2: \{ \ldots P(S1) \ldots V(S1) \ldots \} \]

\[ \tau_4: \{ \ldots P(S1) \ldots V(S1) \ldots \} \]

ready

attempts to lock S1

S1 locked

S1 unlocked

blocked

ready

S1 locked

S1 unlocked

inherits the priority of \( \tau_2 \) after \( \tau_2 \) is blocked
Some Notations

- $J_i$ is the $i$-th job in Task $T$.
- $\pi_i = \text{Assigned priority of Job } J_i$
- $\pi_i(t) = \text{current priority of } J_i$
- If the decision to change the priority of Job $J_i$ is made at $t = t_1$ then
  - $\pi_i(t_1^-)$ = priority at and immediately before,
  - $\pi_i(t_1^+)$ = priority immediately after the priority change

- $\Omega = \text{nonexistent priority, lower than the lowest priority}$
Terminology and Assumptions

- At time $t_1$, job $J_i$ requests resource $R_k$.
- $R_k \rightarrow J_l$: Resource $R_k$ is held by Job $J_l$.
- $J_i \rightarrow R_k$: Job $J_i$ is blocked waiting for resource $R_k$ to be released ($J_i \rightarrow R_k \rightarrow J_l$).

**Scheduling Rules:**
- Ready Jobs scheduled on processors preemptively in a priority driven manner according to their current priorities, $\pi_i(t)$.
- At job release time the priority is equal to its assigned priority.
  - if $J_i$ is release at $t = t'$, then $\pi_i(t') = \pi_i$.

**Resource allocation:**
- If a resource is free then it is allocated when requested.
- if not free then the request is denied and the requesting job is blocked.
Priority Inheritance Rules

- **Scheduling Rule**: same as the assumptions
- **Allocation Rule**: same as the assumptions
- **Priority-Inheritance Rule**:
  - if $J_i \rightarrow R_k \rightarrow J_l$ and $\pi_l(t_1^-) = \text{priority of } J_i \text{ at } t = t_1$
  - then $\pi_l(t_1^+) = \pi_l(t_1)$
  - until $J_l$ releases $R_k$ at $t_2$ when $\pi_l(t_2^+) = \pi_l(t_1^-)$

Properties of Priority Inheritance

- For each resource (semaphore), a list of blocked tasks must be stored in a priority queue.

- A task (job) $\tau_i$ uses its assigned priority, and uses (inherits) the highest dynamic priority of all the tasks it blocks when it is in its critical section and blocks some higher priority tasks.

- Priority inheritance is *transitive*; that is, if task $\tau_i$ blocks $\tau_j$ and $\tau_j$ blocks $\tau_k$, then $\tau_i$ can inherit the priority of $\tau_k$.

- When task $\tau_i$ releases a resource, which priority it should use?

- Chained blocking if requesting multiple resources (nested mutex requests)

- Direct blocking and indirect (inheritance) blocking (when the lower priority task inherits the higher priority task’s priority).
Supplementary Slides