Lecture 10: Scheduling with priorities

Michael O’Boyle
Embedded Software
Overview

• Scheduling dependent tasks
• Mutual exclusion
• Priority Inversion
• Priority Inheritance
  • Deadlock
• Priority Ceiling Protocol
• Summary
Resource access protocols

**Critical sections:** sections of code at which exclusive access to some resource must be guaranteed. Can be guaranteed with semaphores $S$ or “mutexes”.

P($S$) checks semaphore to see if resource is available and if yes, sets $S$ to “used“.
Uninterruptible operations!
If no, calling task has to wait.

V($S$): sets $S$ to “unused“ and starts sleeping task (if any).

Note: Preemption still possible in critical sections
Blocking due to mutual exclusion

Priority $T_1$ assumed to be $>$ than priority of $T_2$.
If $T_2$ requests exclusive access first (at $t_0$), $T_1$ has to wait until $T_2$ releases the resource (at time $t_3$):

For 2 tasks:
blocking is bounded by the length of the critical section

However not true in general
Priority inversion

Priority of $T_1 >$ priority of $T_2 >$ priority of $T_3$.

$T_2$ preempts $T_3$: $T_2$ can prevent $T_3$ from releasing the resource.

Blocking with > 2 tasks can exceed the length of any critical section $T_2$ not involved in critical section but ends up affecting $T_1$
Solution: Forbid preemption in critical sections

Seems a good idea but leads to problems
T1 has high priority but is blocked  T1 independent of lock
Priority inheritance can help

- The idea is that if an important task is blocked by an unimportant one,
  - the unimportant one is elevated and executed quickly to release the lock
- Tasks are scheduled according to their active priorities.
  - Tasks with the same priorities are scheduled.
  - First come first served. As usual
- Rule: tasks inherit the highest priority of tasks blocked by it.
Priority inheritance can help

- **Rule**: tasks inherit the **highest** priority of tasks blocked by it.
  - So if a task $T_1$ executes $P(S)$ & exclusive access already granted to $T_2$, then $T_1$ will become blocked.
  - If $\text{priority}(T_2) < \text{priority}(T_1)$: $T_2$ inherits the priority of $T_1$.
    - $T_2$ resumes.
  - When $T_2$ executes $V(S)$, its priority is decreased to the **highest** priority of the tasks blocked by it.
  - If no other task blocked by $T_2$: $\text{priority}(T_2) :=$ original value.
    Highest priority task so far blocked on $S$ is resumed.
  - **Transitive**: if $T_2$ blocks $T_1$ and $T_1$ blocks $T_0$,
    then $T_2$ inherits the priority of $T_0$. 

Wednesday, 12 February 2014
Priority inheritance in previous example

Previous

\[ T_1 \]

\[ T_2 \]

\[ T_3 \]

\[ P(S) \]

\[ V(S) \]

\[ t \]

Normal execution

Critical section

blocking

\[ P(S) \text{ [sleep]} \]

\[ \text{resume} \]

New

\[ T_1 \]

\[ T_2 \]

\[ T_3 \]

\[ P(S) \]

\[ V(S) \]

\[ t \]

Normal execution

Critical section

\[ T_3 \] inherits the priority of \[ T_1 \] and \[ T_3 \] resumes.
Nested Critical Sections

π: used to denote priority
Transitivity of Priority Inheritance

- $T_1$ blocked by $T_2$ & $T_2$ blocked by $T_3$
- $T_3$ inherits priority from $T_1$

Priority of $T_3$
Priority Inheritance Deadlock

\( \pi T_1 > \pi T_2 \): Priority of \( T_1 \) > \( T_2 \)

\[
\begin{align*}
T_1 &:: \ldots \text{lock}(S_a); \ .a. \text{lock}(S_b); \ .b. \text{unlock}(S_b) \ldots \text{unlock}(S_a); \\
T_2 &:: \ldots \text{lock}(S_b); \ .b. \text{lock}(S_a); \ .b. \text{unlock}(S_a) \ldots \text{unlock}(S_b);
\end{align*}
\]
Priority Ceiling Protocol

• The priority ceiling protocol prevents deadlock and reduces worst case blocking time

• Priority Ceiling (PC) of a resource or semaphore S:
  • PC(S) = highest priority of all processes that may lock S

• A process P is allowed to start a new critical section only if: P’s priority > PC’s of all semaphores locked by processes other than P

• If P is suspended, the process (say, Q) which holds the lock is blocking P
  • Q then inherits P’s priority - execution then follows Priority Inheritance protocol

• A property of this protocol is that any process can be blocked for at most the duration of a single critical section of a lower-priority process
  • A significant gain

• Note assumes fixed known number of tasks and priorities
Example

Consider three processes P1, P2, P3, s.t. \( \pi P_1 > \pi P_2 > \pi P_3 \), with code:

\[
\begin{align*}
P1:: & \text{ begin } \ldots \text{ lock (S1); CS1; unlock(S1); } \ldots \text{ end} \\
P2:: & \text{ begin } \ldots \text{ lock(S1); CS21; lock(S2); CS22; lock(S2); CS23; unlock(S1); } \ldots \text{ end} \\
P3:: & \text{ begin } \ldots \text{ lock(S2); CS3; unlock(S2); } \ldots \text{ end}
\end{align*}
\]

Run through execution sequence starting with just P3 starting first with P2 then P1 entering later

First look at standard priority inheritance

Then look at priority ceiling protocol
Priority Inheritance delays critical task
Priority Ceiling overcomes this

\[-PC(S_1) = \max(\pi_{P_1}, \pi_{P_2}) = \pi_{P_1}\]
\[-PC(S_2) = \max(\pi_{P_2}, \pi_{P_3}) = \pi_{P_2}\]
Walk through of example

- At t0, P3 is ready & starts executing; at t1, P3 locks S2
- At t2, P2 preempts P3 (because $\pi(P2) > \pi(P3)$)
- At t3, P2 attempts to lock S1; however, $\pi(P2) \not> \pi(C(S2))$, which is currently locked by P3
- So, P2 is suspended (not allowed to lock S1), and P3 inherits P2’s priority and continues executing its CS
- At t4, P1 preempts P3 (because $\pi(P1) > \pi(P3)$)
- When P1 attempts to lock S1 sometime later, it secures the lock, because $\pi(P1) > \pi(C(S2))$, the only other semaphore currently locked by another process
- When P1 finishes, P3 resumes, finishes its CS & unlocks S2, at which point, its priority reverts back to $\pi(P3)$
- P2 can then preempt P3 (because now, $\pi(P2) > \pi(P3)$) to obtain S2 & execute its critical section
Priority Ceiling overcomes deadlock

\[ \pi T_1 > \pi T_2: \text{Priority of } T_1 > T_2 \]

T1:: ... lock(Sa); .a. lock(Sb); ... unlock(Sb) ... unlock(Sa);
T2:: ... lock(Sb); .b. lock(Sa); ... unlock(Sa) ... unlock(Sb);

\[ P(Sa) \]

\[ P(Sb) \]
Priority Ceiling Solution

\[ \pi T_1 > \pi T_2: \text{Priority of } T_1 > T_2 \]

T1:: ... lock(Sa); .a. lock(Sb); ... unlock(Sb) ... unlock(Sa);
T2:: ... lock(Sb); .b. lock(Sa); .c. unlock(Sa) .d. unlock(Sb);

\[ P(Sa) \]

\[ V(Sb) \]

\[ \pi \]

\[ T_1 > \pi T_2 \]

\[ P(Sa) \]

\[ a \]

\[ b \]

\[ c \]

\[ d \]
Summary

- Scheduling dependent tasks
- Mutual exclusion
- Priority Inversion
- Priority Inheritance
  - Deadlock
- Priority Ceiling Protocol