Today's topic: Resource Sharing

Priority Inversion and Priority Ceiling Protocols

Basic functions of RTOS kernel

- Time management
- Task management
- Interrupt handling
- Memory management
- Exception handling
- Task scheduling
- Task synchronization
  - Avoid priority inversion
A classic paper on real-time systems


The simplest form of priority inversion
Priority inversion problem

- Assume 3 tasks: A, B, C with priorities Ap<Bp<Cp
- Assume semaphore: S shared by A and C
- The following may happen:
  - A gets S by P(S)
  - C wants S by P(S) and blocked
  - B is released and preempts A
  - Now B can run for a long long period ..... 
  - A is blocked by B, and C is blocked by A 
  - So C is blocked by B
- The above senario is called ‘priority inversion’
- It can be much worse if there are more tasks with priorities in between Bp and Cp, that may block C as B does!

Un-bounded priority inversion
Solutions

- Tasks are ‘forced’ to follow **pre-defined rules** when requesting and releasing resources (locking and unlocking semaphores)
- The rules are called ‘Resource access protocols’
  - NPP, BIP, HLP, PCP

Resource Access Protocols

- Highest Priority Inheritance
  - Non preemption protocol (NPP)
- Basic Priority Inheritance Protocol (BIP)
  - POSIX (RT OS standard) mutexes
- Priority Ceiling Protocols (PCP)
- Immedate Priority Inheritance
  - Highest Locker’s priority Protocol (HLP)
    - Ada95 (protected object) and POSIX mutexes
Non Preemption Protocol (NPP)

- Modify $P(S)$ so that the “caller” is assigned the highest priority if it succeeds in locking $S$
  - Highest priority=non preemption!
- Modify $V(S)$ so that the “caller” is assigned its own priority back when it releases $S$

This is the simplest method to avoid Priority Inversion!

NPP: + and −

- Simple and easy to implement (+), how?
- Deadlock free (++)
- Number of blockings = 1 (+)
- Allow low-priority tasks to block high-priority tasks including those that have no sharing resources (−)

```
Missinig all deadlines!
```

```
[Diagram showing P(s) and V(s) with missing deadlines]
```
Basic Priority Inheritance Protocol (BIP)

- supported in RT POSIX
- **Idea:**
  - A gets semaphore S
  - B with higher priority tries to lock S, and blocked by S
  - B transfers its priority to A (so A is resumed and run with B’s priority)
- **Run time behaviour:** whenever a lower-priority task blocks a higher priority task, it inherits the priority of the blocked task

**Example**

- Task 1: Running with priority H
- Task 2: Using S1
- Task 3: Blocking S2

Diagram:

- H: Blocked
- M: Using S1
- L: Using S2

Legend:
- Red: Blocked
- Yellow: Using S1
- Orange: Using S2
Problem 1: potential deadlock

Task 2: ... P(S2) ... P(S1)...
Task 1: ... P(S1) ... P(S2)...

Problem 2: chained blocking – many preemptions

Task 1 needs M resources may be blocked M times:
→ many preemptions/run-time overheads
→ maximal blocking=sum of all CS sections for lower-priority tasks
BIP: Blocking time calculation

- Let
  - \( CS(k,S) \) denote the computing time for the critical section that task \( k \) uses semaphore \( S \).

- The maximal blocking time for task \( i \):
  - \( B = \sum \{ CS(k,S) \mid \text{task } i, k \text{ share } S \text{ and } \text{pri}(k) < \text{pri}(i) \leq \text{C}(S) \} \)
  - This is not quite true e.g. when there is a deadlock!
    - Deadlock prevention: lock semaphores in increasing order

Properties of BIP: + and -

- Bounded Priority inversion (+)
- Reasonable Run-time performance (+)
- Potential deadlocks (-)
- Chain-blocking – many preemptions (-)
Implementation of Ceiling Protocols

- **Main ideas:**
  - Priority-based scheduling
  - Implement P/V operations on Semaphores to assign task priorities dynamically

Semaphore Control Block for BIP

<table>
<thead>
<tr>
<th>counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
</tr>
<tr>
<td>Pointer to next SCB</td>
</tr>
<tr>
<td>Holder</td>
</tr>
</tbody>
</table>
Standard P-operation (without BIP)

- P(scb):
  
  Disabling interrupt;
  If scb.counter>0 then {scb.counter - 1;
  else
  {save-context();
   current-task.state := blocked;
   insert(current-task, scb.queue);
   dispatch();
   load-context();
  }
  Enable-interrupt

P-operation with BIP

- P(scb):
  
  Disabling interrupt;
  If scb.counter>0 then {scb.counter - 1;
   scb.holder := current-task
   add(current-task.sem-list,scb)}
  else
  {save-context();
   current-task.state := blocked;
   insert(current-task, scb.queue);
   /* queue sorted according to task priority */
   save(scb.holder.priority);
   scb.holder.priority := current-task.priority;
   dispatch();
   load-context();
  }
  Enable-interrupt
Standard V-operation (without BIP)

- V(scb):
  Disable-interrupt;
  If not-empty(scb.queue) then
    { next-to-run := get-first(scb.queue);
      next-to-run.state := ready;
      insert(next-to-run, ready-queue);
      save-context();
      schedule(); /* dispatch invoked*/
      load-context(); }
  else scb.counter ++1;
  Enable-interrupt

V-operation with BIP

- V(scb):
  Disable-interrupt;
  current-task.priority := "original/previous priority"
  /* restore the previous priority of the "caller" i.e current-task*/
  If not-empty(scb.queue) then
    ( next-to-run := get-first(scb.queue);
      /*queue sorted according to task priority*/
      next-to-run.state := ready;
      scb.holder := next-to-run;
      add(next-to-run.sem-list, scb);
      insert(next-to-run, ready-queue);
      save-context();
      schedule(); /* dispatch invoked*/
      load-context(); }
  else scb.counter ++1;
  Enable-interrupt
Immediate Priority Inheritance:
= Highest Locker’s Priority Protocol (HLP)
= Priority Protect Protocol (PPP)

- Adopted in Ada95 (protected object), POSIX mutexes
- **Idea:** define the ceiling $C(S)$ of a semaphore $S$ to be the highest priority of all tasks that use $S$ during execution. Note that $C(S)$ can be calculated statically (off-line).

Run-time behaviour of HLP

- Whenever a task succeeds in holding a semaphore $S$, its priority is changed dynamically to the maximum of its current priority and $C(S)$.
- When it finishes with $S$, it sets its priority back to what it was before
Example

<table>
<thead>
<tr>
<th>priority</th>
<th>use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task 1</td>
<td>H S3</td>
</tr>
<tr>
<td>Task 2</td>
<td>M S1, S</td>
</tr>
<tr>
<td>Task 3</td>
<td>L S1, S2</td>
</tr>
<tr>
<td>Task 4</td>
<td>Lower S2, S</td>
</tr>
</tbody>
</table>

C(S1)=M  
C(S2)=L  
C(S3)=H  
C(S)=M

Example: Highest Locker’s Priority Protocol

M and Lower share S

- Computing
- Blocked
- Using resource
Property 1: Deadlock free (HLP)

Once task 2 gets S1, it runs with pri H, task 1 will be blocked (no chance to get S2 before task 2)

Property 2:
Tasks will be blocked at most once
HLP: Blocking time calculation

Let
- \( CS(k,S) \) denote the computing time for the critical section that task \( k \) uses semaphore \( S \).

Then the maximal blocking time \( B \) for task \( i \) is as follows:
- \( B = \max\{CS(k,S) | \text{ task } i, k \text{ share } S, \ pri(k)<\text{pri}(i)\leq C(S)\} \)

Implementation of HLP

- Calculate the ceiling for all semaphores
- Modify SCB
- Modify P and V-operations
Semaphore Control Block for HLP

<table>
<thead>
<tr>
<th>counter</th>
</tr>
</thead>
<tbody>
<tr>
<td>queue</td>
</tr>
<tr>
<td>Pointer to next SCB</td>
</tr>
<tr>
<td>Ceiling</td>
</tr>
</tbody>
</table>

P-operation with HLP

- P(scb):
  
  Disable-interrupt;
  If scb.counter>0 then
  { scb.counter - 1;
    save(current-task.priority);
    current-task.priority := Ceiling(scb) }
  else
  {save-context();
   current-task.state := blocked
   insert(current-task, scb.queue);
   dispatch();
   load-context() }
  Enable-interrupt
V-operation with HLP

- V(scb):
  
  Disable-interrupt;
  
  current-task.priority := get(previous-priority)

If not-empty(scb.queue) then
  
  next-to-run := get-first(scb.queue);
  
  next-to-run.state := ready;
  
  next-to-run.priority := Ceiling(scb);
  
  insert(next-to-run, ready-queue);
  
  save-context();
  
  schedule(); /* dispatch invoked*/
  
  load-context();

end then
else scb.counter ++1;
end else

Enable-interrupt

Properties of HLP: + and -

- Bounded priority inversion
- Deadlock free (+), Why?
- Number of blocking = 1 (+), Why?
- HLP is a simplified version of PCP (+)
- The extreme case of HLP=NPP (-)
  - E.g. when the highest priority task uses all semaphores, the lower priority tasks will inherit the highest priority
Summary

<table>
<thead>
<tr>
<th></th>
<th>NPP</th>
<th>BIP</th>
<th>HLP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounded Priority Inversion</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Avoid deadlock</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Avoid Un-necessary blocking</td>
<td>no</td>
<td>yes</td>
<td>yes/no</td>
</tr>
<tr>
<td>Blocking time calculation</td>
<td>Easy</td>
<td>hard</td>
<td>easy</td>
</tr>
</tbody>
</table>

Priority Ceiling Protocol (combining HLP and BIP)

- Each semaphore $S$ has a Ceiling $C(S)$
- **Run-time behaviour**:
  - Assume that $S$ is the semaphore with highest ceiling locked by other tasks currently: $C(S)$ is “the current system ceiling”
  - If $A$ wants to lock a semaphore (not necessarily $S$), it must have a **strictly higher** priority than $C(S)$ i.e. $P(A) > C(S)$. Otherwise $A$ is blocked, and it transmits its priority(+e) to the task currently holding $S$
Example: PCP

A: \[\ldots P(S_1)\ldots V(S_1)\ldots\]  \quad \text{Prior(A)=H}
B: \[\ldots P(S_2)\ldots P(S_3)\ldots V(S_3)\ldots V(S_2)\ldots\]  \quad \text{Prior(B)=M}  \quad C(S_1)=H
C: \[\ldots P(S_3)\ldots P(S_2)\ldots V(S_2)\ldots V(S_3)\]  \quad \text{Prior(C)=L}  \quad C(S_2)=C(S_3)=M

PCP:  Blocking time calculation

- Let
  - \(CS(k,S)\) denote the computing time for the critical section that task \(k\) uses semaphore \(S\).

- The maximal blocking time for task \(i\):
  - \(B = \max\{CS(k,S) \mid \text{ task } i,k \text{ share } S, \text{ pri}(k)<\text{pri}(i)\leq C(S)\}\)

(which is the same as HLP)
Exercise: implementation of PCP

- Implement P,V-operations that follow PCP
- (this is not so easy)

Properties of PCP: + and -

- Bounded priority inversion (+)
- Deadlock free (+)
- Number of blocking = 1 (+)
- Better response times for high priority tasks (+)
  - Avoid un-necessary blocking
- Not easy to implement (-)
## Summary

<table>
<thead>
<tr>
<th></th>
<th>NPP</th>
<th>BIP</th>
<th>HLP</th>
<th>PCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bounded Priority Inversion</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Deadlock free</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Un-necessary blocking</td>
<td>yes</td>
<td>no</td>
<td>yes/no</td>
<td>no</td>
</tr>
<tr>
<td>Blocking time calculation</td>
<td>easy</td>
<td>hard</td>
<td>easy</td>
<td>easy</td>
</tr>
<tr>
<td>Number of blocking</td>
<td>1</td>
<td>&gt;1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Implementation</td>
<td>easy</td>
<td>easy</td>
<td>easy</td>
<td>hard</td>
</tr>
</tbody>
</table>