Today’s topic: Resource Sharing

Priority Inversion and Priority Ceiling Protocols

1

Basic functions of RTOS kernel

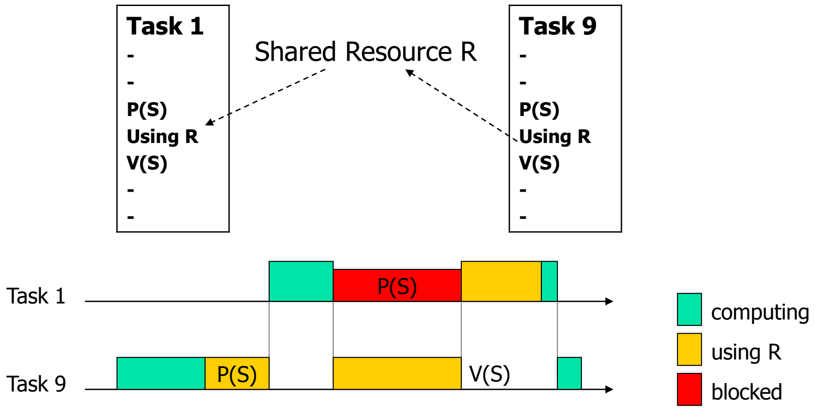
- Time management
- Task mangement
- Interrupt handling
- Memory management
- Exception handling
- Task scheduling
- Task synchronization
  - Avoid priority inversion

2

A classic paper on real-time systems

- L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization. In *IEEE Transactions on Computers*, vol. 39, pp. 1175-1185, Sep. 1990.

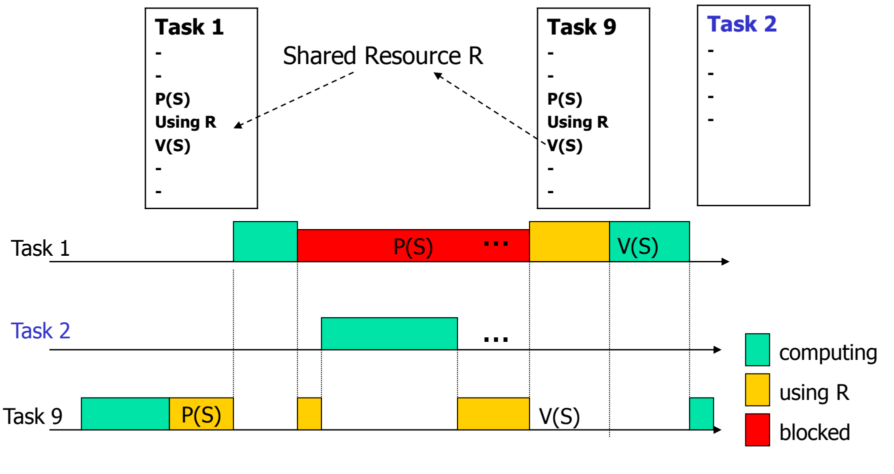
The simpliest form of priority inversion



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)

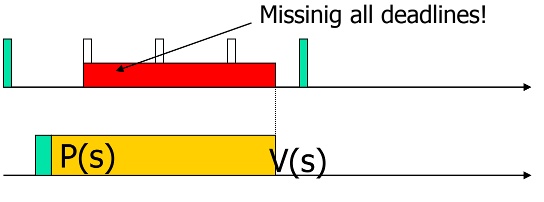
- Modify P(S) so that the "caller" is assigned the highest priority if it succeeds in locking S
  - Highest priority=non preemtion!
- Modify V(S) so that the "caller" is assigned its own priority back when it releases S

This is the simplest method to avoid Priority Inversion!

9

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 (-)

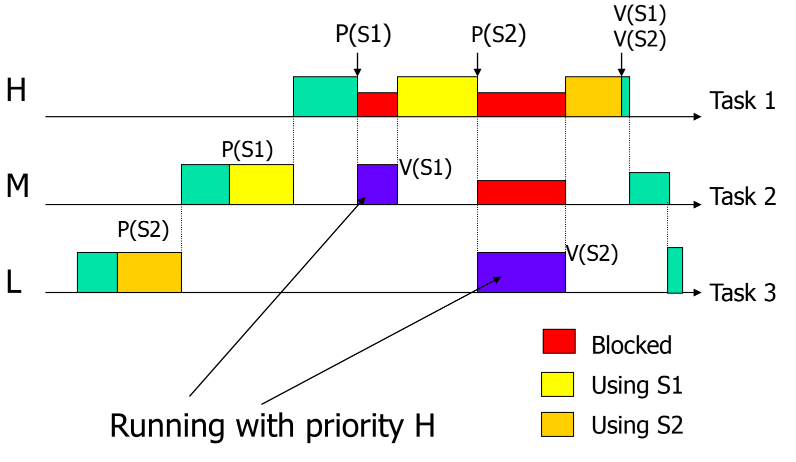


10

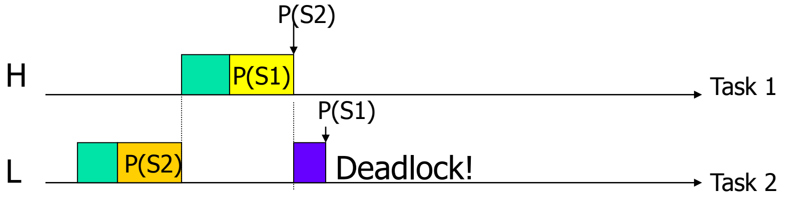
Basic Priority Inheritance Protocol (BIP)

- 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-priotity task blocks a higher priority task, it inherits the priority of the blocked task

Example



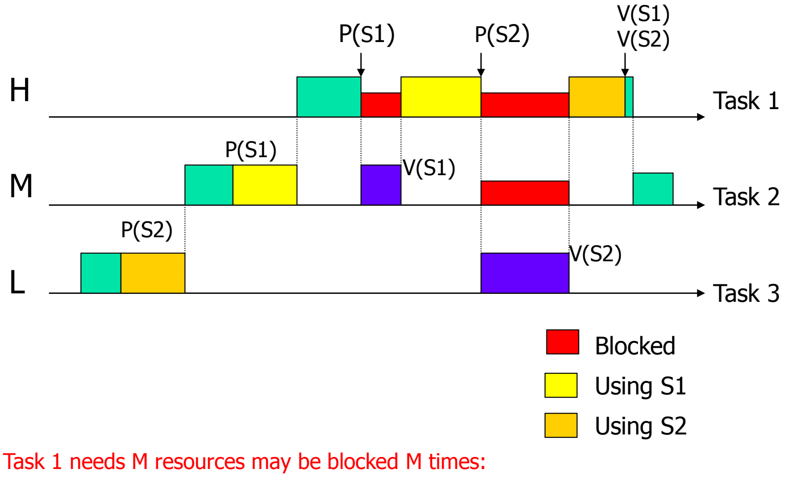
Problem 1: potential deadlock



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

13

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

14

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= ∑{CS(k,S)| task i, k share S and pri(k)<pri(i)<=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

counter
queue
Pointer to next SCB
Holder

Standard P-operation (without BIP)

- P(scb):

```
Disable-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):

```
Disable-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.priotiry);
   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
```

21

V-operation with BIP

- V(scb):

```
 Disable-interrupt;
 current-task.priority := "original/previous priority"
    /* restore the previous priority of the "caller"  i.e current-tast*/
    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
```

22

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

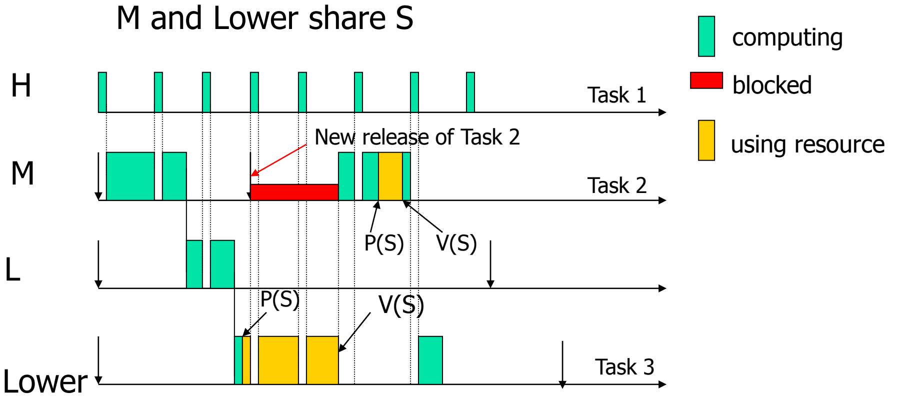
- Whenever a task succeeds in holding a semaphor S, its priority is changed dynamically to the maximum of its current priority and C(S).
- When it finishes with S, it sets its priority back to what it was before

Example

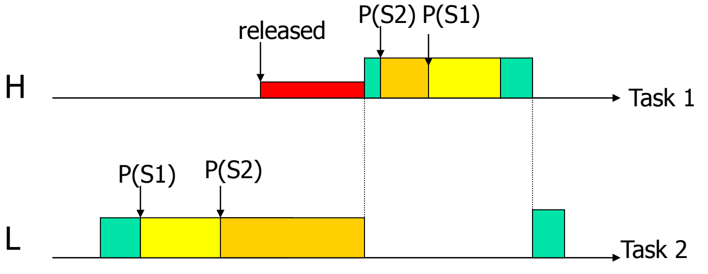
	priority	use
Task 1	H	S3
Task 2	M	S1, S
Task 3	L	S1, S2
Task 4	Lower	S2, S

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

Example: Highest Locker's Priority Protocol

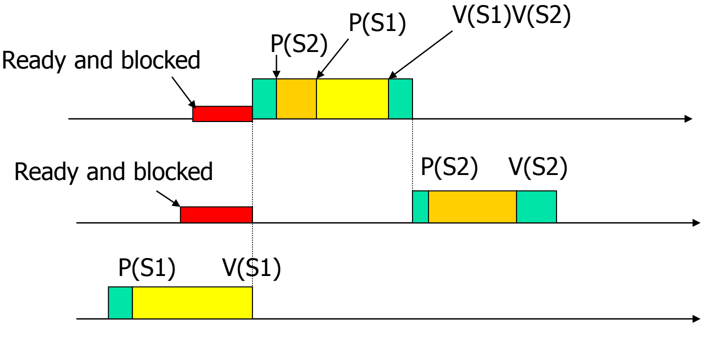


Property 1: Deadlock free (HLP)



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)| task i,k share S, pri(k)<pri(i)<=C(S)}

Implementation of HLP

- Calculate the ceiling for all semaphores
- Modify SCB
- Modify P and V-operations

Semaphore Control Block for HLP

counter
queue
Pointer to next SCB
Ceiling

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

	NPP	BIP	HLP
Bounded Priority Inversion	yes	yes	yes
Avoid deadlock	yes	no	yes
Avoid Un-necessary blocking	no	yes	yes/no
Blocking time calculalation	Easy	hard	easy

Priority Ceiling Protocol (combining HLP and BIP)

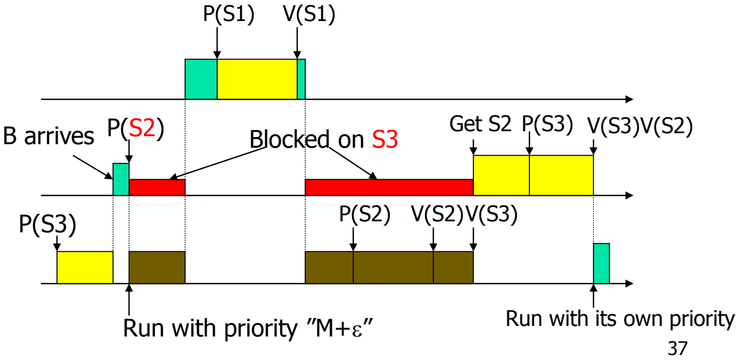
- Each semaphore S has a Ceiling C(S)
- Run-time behaviour:
  - Assume that S is the semaphore with highest ceiling locked by other tasks currently: C(S) is "the current system ceiling"
  - If A wants to lock a semaphore (not necessarily S), it must have a strictly higher priority than C(S) i.e. $P(A)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

Example: PCP

A: ...P(S1)...V(S1)...
B: ...P(S2)...P(S3)...V(S3)...V(S2)...
C: ...P(S3)...P(S2)...V(S2)...V(S3)

Prio(A)=H
Prio(B)=M
Prio(C)=L

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



37

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)| task i,k share S, pri(k)<pri(i)<=C(S)}

(which is the same as HLP)

38

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

	NPP	BIP	HLP	PCP
Bounded Priority Inversion	yes	yes	yes	yes
Deadlock free	yes	no	yes	yes
Un-necessary blocking	yes	no	yes/no	no
Blocking time calculalation	easy	hard	easy	easy
Number of blocking	1	>1	1	1
Implementation	easy	easy	easy	hard

41