Resource Access Protocols

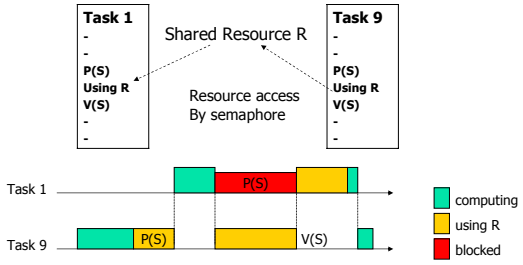
The Blocking Problem

- Now we remove the assumption that tasks are independent, sharing no resources, not interacting
- The main problem: priority inversion
  - High priority tasks are blocked by lower priority tasks
- The main sources of priority inversion:
  - Non preemptable sections (we mentioned this earlier)
  - Sharing resources
  - Synchronization and mutual exclusion
- The response time calculation should be modified

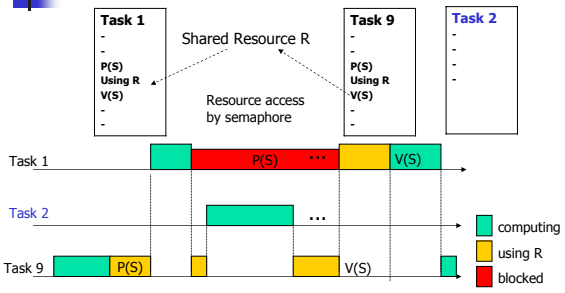
Contents

- Priority inversion phenomenon
- Resource access protocols
  - Highest Priority Inheritance
    - Non preemption protocol (NPP)
  - Immedate Priority Inheritance
    - Highest Locker's priority Protocol (HLP)
      - Ada95 (protected object) and POSIX mutexes
  - Basic Priority Inheritance Protocol (BIP)
    - POSIX (RT OS standard) mutexes
  - Priority Ceiling Protocols (PCP)
- Blocking and response time analysis

The simpliest form of priority inversion



Un-bounded priority inversion



Solutions

- Tasks are 'forced' to follow certain rules when locking and unlocking semaphores (requesting and releasing resources)
- The rules are called 'Resource access protocols'
  - NPP, BIP, HLP, PCP
- A common rule of these protocols: tasks must release all semaphores before the next instance; when a task finishes executing, it must have released all semaphores

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!

- The equation for response time calculation:

$$Ri = Bi + Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$$

  - Where Bi is the longest time that task i can be blocked by lower-priority tasks with non preemptive section

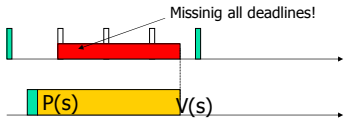
Exercise

(1)Show the run-time schedules with(out) NPP
(2) Calculate the worst case response times
(note that R1=85)

	C	T	Blocking
T1	20	80	10
T2	30	110	0
T3	70	200	65

NPP: + and –

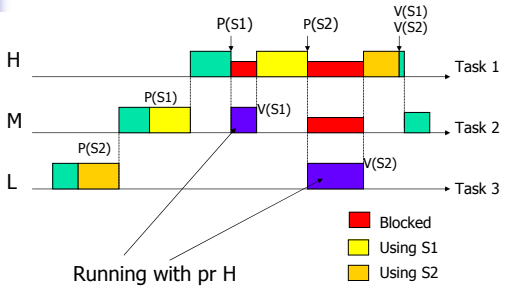
- Simple and easy to implement (+), how?
- Deadlock free (++), why?
- Number of blocking = 1 (+), Why?
- Allow low-priority tasks to block high-priority tasks including those that have no sharing resources (-)



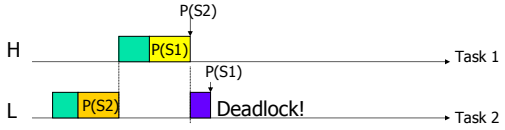
Basic Priority Inheritance Protocol (BIP)

- 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

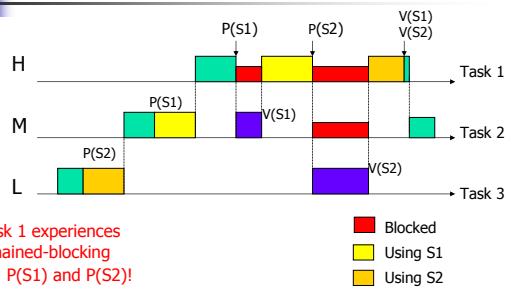


Property/Problem 1: potential deadlock



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

Property/Problem 2: chained blocking
(H needs M resources may be blocked M times)



Task 1 experiences Chained-blocking by P(S1) and P(S2)!

SCB to implement BIP

- Semaphores Control Block for PIP

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
```


V-operation with BIP

- V(scb):

```
Disable-interrupt;
current-task.priority := "original/prvious priority"
    /* highest-priority of tasks blocked by smaphors ownd by current-task*/
    /* check all blocked tasks waiting for sem in current-tcb.sem-list*/
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
```

Properties of BIP: + and -

- Bounded Priority inversion (+)
- Reasonable Run-time performance (+)
- Require no info on resource usage of tasks (+)
- potential deadlock, the same reason as in OS (-)
- It may cause chain-blocking (a task needs M semaphores may be blocked M times!): why, example? (-)
- Complicated to compute the maximal blocking times and response times (-)

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

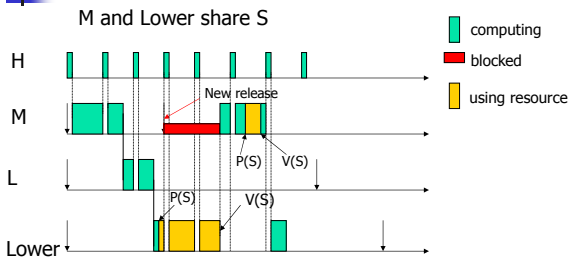
- Whenever a task succeeds in holding a semaphor S, its priority is changed dynamically to the maximum of its current priority and C(S).
- When it finishes with S, it sets its priority back to what it was before

Example

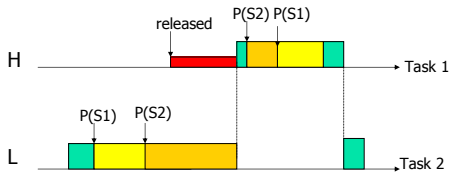
	priority	use
Task 1	H	S3
Task 2	M	S1, S
Task 3	L	S1, S2
Task 4	Lower	S2, S

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

Example: Highest Locker's Priority Protocol

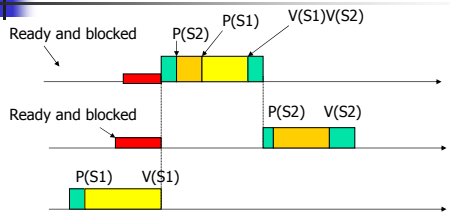


Property 1: Deadlock free (HLP)



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: Response Time Analysis

- Let
  - CS(k,S) denote the computing time for the critical section that task k uses semaphore S.
  - Use(S) is the set of tasks using S
- Then the maximal blocking time B and response time Ri for task i is as follows:
  - B =max{CS(k,S)| i,k in Use(S), pr(k)<pr(i)<=C(S)}
  - $Ri = B + Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$

Implementation of HLP

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

SCB to implement HLP

- Semaphores 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 := C(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 := C(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 (priority ceiling protocol, with the same worst case blocking time as PCP (+)
- The extreme case of HLP=NPP (-)
  - E.g when the highest priority task uses all semaphores (even only once in its whole life), 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) = the priority of the highest priority task that can lock S
- Run-time behaviour:
  - Assume that S is the semaphore with highest ceiling locked by other tasks currently
  - If a task A wants to lock a semaphore (not necessarily S), it must have a strictly higher priority than the ceilings of all semaphors locked by other tasks, i.e. P(A) > C(S). Otherwise A is blocked, and it transmitts its priority(+ε) to the task currently holding S

PCP: Response Time Analysis (precisely the same as HLP)

- Let
  - CS(k,S) denote the computing time for the critical section that task k uses semaphore S.
  - Use(S) is the set of tasks using S
- Then the maximal blocking time B and response time Ri for task i is as follows:
  - B=max{CS(k,S)| i,k in Use(S), pr(k)<pr(i)<=C(S)}
  - $Ri = B + Ci + \sum_{j \in HP(i)} \lceil Ri/Tj \rceil * Cj$

Example: PCP

	C	D	T	Priority
Task A	5	10	50	H
Task B	250	500	500	M
Task C	1000	3000	3000	L

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

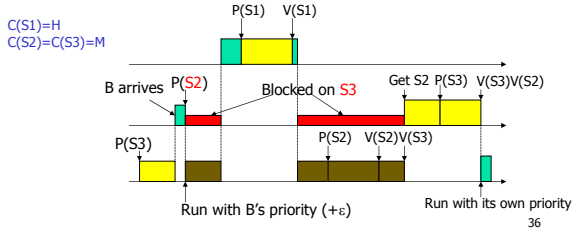
Thus:
C(S1)=H
C(S2)=C(S3)=M

Example: PCP

	C	D	T	Priority
Task A	5	10	50	H
Task B	250	500	500	M
Task C	1000	3000	3000	L

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

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



Exercise: implementation of PCP

- Implement P,V-operations that follow PCP

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
Avoid deadlock	yes	no	yes	yes
Avoid Un-necessary blocking	no	yes	yes/no	yes
Blocking time calculalation	easy	hard	easy	easy
Number of blocking	1	>1	1	1
Implementation	easy	easy	easy	hard