Priority inversion

Suppose
• low-priority task locks resource;
• it is preempted by long running middle-priority task
  → blocks high-priority task that needs resource

Solution: **Priority inheritance**:  
• A task that holds a resource via a semaphore  
  executes at the priority of highest-priority task  
  that is blocked on that resource

Avoiding priority inversion and deadlock

**Priority ceiling** (for dynamic priorities):  
• For every shared resource, compute ceiling:
  priority of highest priority task that may lock resource

At runtime:
• Task may lock resource only if  
  task priority > ceiling of all currently locked resources

• Task that holds lock inherits highest priority of blocked  
  tasks on that lock until release of lock  
  [priority inheritance]

Example Priority Ceiling (1)

Consider 5 tasks and two resources A and B  
• Task 1 needs A and B (in this order), prio 6  
• Task 2 needs B and A (in this order), prio 5  
• Task 3, executes long, priority 3  
• Task 4 needs A, priority 2  
• Task 5 needs B, priority 1  
(higher number = higher priority)

Example Priority Ceiling (2)

Consider 5 tasks and two resources A and B  
• Task 1 needs A and B, prio 6  
• Task 2 needs B and A, prio 5  
• Task 3 executes long, prio 3  
• Task 4 needs A, prio 2  
• Task 5 needs B, prio 1

Without any inheritance/ceiling protocol:
• Task 5 arrives, gets B  
• Task 4 arrives, preempt task 5 and gets A  
• Task 3 arrives and gets blocked  
• Task 3 arrives, preempt task 4, executes long, terminates  
• Task 4 resumes, releases resource A  
• Task 5 resumes, releases resource B  
• Tasks 2 resumes, gets B  
• Task 1 arrives, preempt task 2 and gets A, but later blocks  
  on B → deadlock
Example Priority Ceiling (3)

- Task 1 needs A and B, prio 6
- Task 2 needs A and B, prio 5
- Task 3 executes long, prio 3
- Task 4 needs A, prio 2
- Task 5 needs B, prio 1

With priority ceiling:
- A has ceiling 6, B has ceiling 6
- Task 5 gets B
- Task 4 blocks (prio A=2 < 6=ceiling B)
- Task 2 arrives, gets blocked, task 5 gets prio 5
- Task 3 arrives, gets blocked
- Task 5 completes and releases B
- Task 2 gets B
- Task 1 arrives, gets blocked when it needs A because its prio is not greater than all locked resources (B has also prio 6)
- Task 2 gets A, completes and releases A and B
- Task 1 executes and completes
- Task 3 executes long and completes
- Task 4 executes and completes

Comparison

Without priority ceiling
- Task 5 arrives, gets B
- Task 4 arrvies, preempts 5 and gets A
- Task 2 arrives, gets blocked
- Task 3 arrives, preempts 4, executes long, completes
- Task 4 completes, releases A
- Task 3 completes, releases B
- Task 2 resumes, gets B
- Task 1 arrives, preempts 2 and gets A, but later blocks on B

With priority ceiling
- Task 5 completes, releases B
- Task 2 gets B
- Task 1 arrives, gets blocked when it needs A because its prio is not greater than all locked resources (B has also prio 6)
- Task 2 gets A, completes, releases A and B
- Task 1 executes and completes
- Task 3 executes long and completes
- Task 4 executes and completes

Xenomai #10a
- Measure jitter of periodic scheduling
  [Try with VMWare/VirtualBox]

Delivery program for Linux PC

Xenomai #10b
- Use Excel to analyze the results
  [Sufficient to deliver results of Linux PC (not VMware/VirtualBox ...)]

Xenomai #10d
- Measure interrupt latency by using special cable which connects two PCs:
  - It connects D0 of one PC to interrupt line S6 of other and vice versa
- Enable interrupts on parallel port
- PCI & PC2 first both set D0 to high [set ioperm]
  - outb(0x378) & 0x01, 0x378; outb(inb(0x378) & 0xfe, 0x378);
- PC1 measures time (now = rt_timer_read()) and set D0 low and high again [investigate the best moment to measure time]
  - PC1 interrupt handler measures time

Assignment for 7 October 2015

Xenomai exercise #10
- Partly: not VMware results, not extra load 10c
  - 10a: deliver Linux PC program
  - 10b: deliver results of Linux PC (not VMware/..)
  - 10c: NOT
  - 10d: deliver program and results

Mail results before Tuesday 6 October 18:00