Priority Inheritance and Priority Ceiling Protocols


Slides adopted from CS554, Spring 2009 at SUNY, Binghampton http://www.cs.binghamton.edu/~kang/teaching/cs554/
Priority Inversion

- A high priority task is blocked due to a low priority task

- How can it happen?
  - Mutex for shared resource access
  - Non-preemptive subsystem access
    - Network
    - System bus
    - Secondary storage
Mutual Exclusion via Semaphores

- Ensure only one task/process is in the critical section
  - `wait(S)` to get access to semaphore S
  - `signal(S)` to release S
  - Example: producer consumer

```c
Producer()
{
    
    
    wait(S)
    critical section /* create data & increment pointer */
    signal(S)
    
    
}
```
Priority Inversion

Task-1 is of higher priority than Task-4
Orange section signifies execution with Resource R shared between T1 and T4

Source of the figure: Chenyang Lu, Washington University Saint Louis
Unbounded Priority Inversion
What really happened on Mars?

Repeated resets in the Mars Pathfinder

- The Mars Pathfinder mission was widely proclaimed as "flawless" in the early days after its July 4th, 1997 landing on the Martian surface.... But a few days into the mission, not long after Pathfinder started gathering meteorological data, the spacecraft began experiencing total system resets, each resulting in losses of data. The press reported these failures in terms such as "software glitches" and "the computer was trying to do too many things at once"....

- For a full story, visit http://research.microsoft.com/%7Embj/Mars_Pathfinder/Mars_Pathfinder.html
Pathfinder Incident

- Classical priority inversion problem due to shared system bus!

Source of the figure: Damir Isovic, Malardaren University, Sweden
Priority Inheritance

- Inherit the priority of the blocked high priority task
Priority Inheritance Protocol (PIP)

- If $T_L$ blocks a higher priority task $T_H$, $\text{priority}(T_L) \leftarrow \text{priority}(T_H)$
- When $T_L$ releases a semaphore:
  - Return to its normal priority if it doesn’t block any task
  - Otherwise, set $\text{priority}(P_L) \leftarrow$ highest priority of the tasks blocking on a semaphore held by $T_L$
- Transitive
  - $T_1$ blocked by $T_2$: $\text{priority}(T_2) \leftarrow \text{priority}(T_1)$
  - $T_2$ blocked by $T_3$: $\text{priority}(T_3) \leftarrow \text{priority}(T_1)$
In the worst case, the highest priority task $T_1$ can be blocked by $N$ lower priority tasks in the system when $T_1$ has to access $N$ semaphores to finish the execution!
For RMS schedulability test, ensure that:

\[ \sum_{i=1, N} \left( \frac{C_i}{T_i} + \frac{B_i}{T_i} \right) \leq n(2^{1/n} - 1) , \ 1 \leq i \leq n \]

- \( B_i \) is the the longest priority inversion time that could be experienced by \( T_i \)
- \( B_i = CS_i + CS_{i+1} + \ldots + CS_k \) where \( CS_i, CS_{i+1}, \ldots, CS_k \) are the critical sections by which lower priority tasks could block \( T_i \)
- **Sufficient but not necessary** because the priority due to contention for \( CS \) may not happen in reality
Priority Ceiling Protocol (PCP)

- Avoid chained blocking
  - Guarantee a task is blocked by at most one lower priority task
  - No deadlock

- Assumptions
  - Each task is associated with a fixed priority
PCP

- Each semaphore has the fixed priority ceiling
  - $\text{ceiling}(S) =$ highest priority among all the tasks that will request $S$

- How it works:
  - $T_i$ can access a semaphore $S$ if
    - $S$ is not already allocated to any other task; and
    - Priority of $T_i$ is higher than the current processor ceiling
      - $\text{ceiling} = \max(\text{priority ceilings of all the semaphores allocated to tasks other than } T_i)$
Ceiling(S1) = 1 (high)
Ceiling(S2) = 2 (medium)
Ceiling(S3) = 2 (medium)

No chained blocking!

Source of the figure: Damir Isovic, Malardaren University, Sweden
Schedulability of RMS with PCP

- Consider blocking time:

\[ \sum_{i=1, N} \left( \frac{C_i}{T_i} + \frac{B_i}{T_i} \right) \leq n\left(2^{1/n} - 1\right), \quad 1 \leq i \leq n \]

- \( B_i = \max(CS_i, CS_{i+1}, ..., CS_k) \) where \( CS_i, CS_{i+1}, ..., CS_k \) are the critical sections that can block \( T_i \)