Priority Inversion
Roadmap for Today

- Priority Inversion
  - Resource protocols

- Synchronization Mechanisms
  - Signaling
  - Semaphores
  - Monitors

- Synchronization Problems
  - Producer / Consumer and
  - Reader / Writer
Example

- 5 processes
  - process number equals priority
  - Priority of P5 > priority of P1
  - Release and execution times as shown
  - No deadlines (only an example for later comparison)
  - Priority-driven preemptively scheduled
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example
Example
Reality is more complex

- Processes are not usually independent
Real-Time Traffic Scheduling

- Two process streams
- A high priority & a low priority
Problem

- Intersection is a *mutually exclusive resource*
Mutual Exclusion

- Can be solved by resource access protocols
Priorities and Resource Contention

Main Reference
Pane W. S. Liu “Real-time Systems”, Chapter 8
Resources

- Processes require resources in order to execute. (e.g. locks, ports, memory, ...)

- Resource characteristics
  - Serially reusable,
  - Mutually exclusive

- We ignore resources that
  - are infinitely available or exceed demand,
  - or can be pre-allocated.
Resource Contention Problem

- Priority inversion.
  - We need to, at least, bound the length of priority inversion.
  - Preferably minimize the length of priority inversion.

Famous example of priority inversion:

**Mars Pathfinder 1997**
Marth Pathfinder

Mars Path Finder and the famous Mars Rock YOGI
Resource Contention Problems

- Timing anomaly
- Deadlock
Major Assumption

- Single processor system
Our Example + 2 Resources

Resource 1
Resource 2

Nested Critical Section*

*P2 first needs R1 and then later additionally R2
Simple Priority Driven Scheduling
Example

SPD Scheduling
Example

SPD Scheduling
Example

SPD Scheduling

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

SPD Scheduling
Example

SPD Scheduling
Example
Example
Example

SPD Scheduling
Example

SPD Scheduling
Example

SPD Scheduling
Example
Example

SPD Scheduling

P5

P4

P3

P2

P1

0 2 4 6 8 10 12 14 16 18 20
Result

- The most important processes P5 and P4 are heavily delayed
- P3 is almost not delayed due to its characteristic, it does not need any resource

⇒ Find a better solution
4 Resource Allocation Protocols

- Non Preemptive Critical Sections (NPCS)
- Priority Inheritance (PI)
- Priority-Ceiling Protocol (PCP)
- Stacked Priority-Ceiling Protocol (SPCP)
- ... and some others
  - See text book (Liu)
Nonpreemptive Critical Sections

- As soon as a process holds a resource it is **no longer preemptable**
  - Prevents deadlock
  - Bounds priority inversion
    - Max blocking time is the **maximum execution time** of the critical sections of all lower priority processes

*This process gets **highest priority** in system
Non-Preemptive Critical Sections
Example

NPCS Scheduling
Example

NPCS Scheduling
Example

NPCS Scheduling
Example

NPCS Scheduling
Example

NPCS Scheduling
Example
Example

NPCS Scheduling

P1
P2
P3
P4
P5
Example

NPCS Scheduling

<table>
<thead>
<tr>
<th>P5</th>
<th>P4</th>
<th>P3</th>
<th>P2</th>
<th>P1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

0  2  4  6  8  10  12  14  16  18  20
Example

NPCS Scheduling
Example

NPCS Scheduling
Example

NPCS Scheduling

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

NPCS Scheduling
Example

NPCS Scheduling

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

NPCS Scheduling

0  2  4  6  8  10  12  14  16  18  20
Example

NPCS Scheduling

P1
P2
P3
P4
P5
Example

NPCS Scheduling
Comparison with SPD-Scheduling
Analysis: Nonpreemptive Critical Sections

- **Pros**
  - Simple
  - No prior knowledge of resource requirements needed
  - Prevents deadlock

- **Cons**
  - Low priority process blocks high priority process even when there are no resource conflicts
  - Protocol only suitable for trusted software
    - Usually implemented by interrupt disabling
  - In CS there is no system calls otherwise CPU wasting in case of a “blocking” system call
Worst-Case Blocking Time

- Longest lower-priority critical section:

\[ bt_i(rc) = \max_{i+1 \leq k \leq n} \{cst_k\} \]

- \( bt \) = blocking time
- \( cst \) = critical section time

Not that realistic
Priority Inheritance (PI)

- When a high-priority process (P3) blocks, the low-priority process (P1) inherits the current priority of the blocking process.

- PI bounds priority inversion.
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance

PI Scheduling
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance

PI Scheduling
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance
Example with Priority Inheritance

PI Scheduling

P5
P4
P3
P2
P1

0 2 4 6 8 10 12 14 16 18 20

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example with Priority Inheritance

PI Scheduling
Example with Priority Inheritance
Example with Priority Inheritance
Comparison with SPD Rule
Analysis: Priority Inheritance

- **Pros**
  - Prevents uncontrolled priority inversion.
  - Needs no knowledge of resource requirements.

- **Cons**
  - Does not prevent deadlock.
  - Does not minimise blocking times.
    - With chained blocking, worst-case blocking time is \( \min(n,m) \) critical sections
      - \( n \) = number of lower priority processes that can block \( P \)
      - \( m \) = number of resources that can be used to block \( P \)
  - Some overhead in a **release** or **acquire** operation
Chained Blocking

- 4 lower priority processes
- 4 potentially conflicting resources
- Worst-case blocking time $= 16$ units\(^1\)

\(^1\)Assume lower priority process allocates its first resource just before higher priority process runs
Priority Ceiling Protocol

- Avoids deadlock by defining an order of resource acquisition
- Prevents transitive (chained) blocking
  - Worst-case blocking time = single critical section

Description how to implement PCP, see:
http://www.awprofessional.com/articles/article.asp?p=30188&seqNum=5&rl=1
Priority Ceilings

- Resources required by all processes are known a priori
  - Similar approach as with deadlock avoidance

- Priority ceiling of resource $R_i$ is equal to the highest priority of all processes that use $R_i$

- Priority ceiling of system is highest priority ceiling of all resources currently in use
Priority Ceilings of Our Example

Priority Ceilings of Ri

PCP Scheduling
Priority Ceiling Protocol Rules

- Priority inheritance applies as before.
- When a process (P) requests a resource (R) either:
  - If R is *allocated* ⇒ P blocks (+ priority inheritance)
  - If R is *free*,
    - If P’s *current priority > system’s priority ceiling* ⇒
      R is allocated to process P
    - If P’s current priority ≤ system’s priority ceiling ⇒
      P blocks – except if:
      - P already holds a resource whose priority ceiling is equal to the system’s priority ceiling
Example

PCP Scheduling

- P5
- P4
- P3
- P2
- P1

0 2 4 6 8 10 12 14 16 18 20

max
Example

PCP Scheduling

P1

P2

P3

P4

P5
Example

PCP Scheduling

P1
P2
P3
P4
P5
curr

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

PCP Scheduling

Prio(P2) < CurrSPC $\Rightarrow$ no allocation
Example

PCP Scheduling

... but P1 inherits prio(P2) = 2
Example

PCP Scheduling
Example

PCP Scheduling
Example

PCP Scheduling

P5
P4
P3
P2
P1

0  2  4  6  8  10  12  14  16  18  20
Example

PCP Scheduling
Example

PCP Scheduling
Example
Example

PCP Scheduling
Example

PCP Scheduling

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

PCP Scheduling

0  2  4  6  8  10  12  14  16  18  20

P1
2  4

P2

P3

P4

P5
Example

PCP Scheduling
Example
Example

PCP Scheduling
Comparison to Previous Example

PCP Scheduling

P5
P4
P3
P2
P1

© 2009 Universität Karlsruhe (TH), System Architecture Group
Analysis: Priority Ceiling Protocol

- **Pros**
  - Avoids deadlocks
  - If a process doesn’t self suspend, a process is *blocked at most once* during execution
  - Processes cannot be transitively blocked
    - ⇒ minimizes blocking time to the longest lower-priority conflicting critical section (+ context switches)
    - Processes only receive their first resource when all required resources are not held by lower priority processes

- **Cons**
  - *A priori knowledge* of resource needs is required
Stack-Based Priority Ceiling Protocol

- The motivation is to share a single stack for all processes
  - Saves stack space.

- Restriction: processes cannot self-suspend.
Rules

- **Scheduling:**
  - After a process is released, it is blocked from starting until its assigned priority is higher than the current system priority ceiling.
  - Unblocked processes are preemptively priority scheduled according to their assigned priority.

- **Resource allocation:**
  - Whenever a process requests a resource it receives the resource.
Example

SPCP Scheduling
Example

SPCP Scheduling
Example
Example

SPCP Scheduling
Example

SPCP Scheduling

P5

P4

P3

P2

P1

0  2  4  6  8  10  12  14  16  18  20

© 2009 Universität Karlsruhe (TH), System Architecture Group
Example

SPCP Scheduling

P5

P4

P3

P2

P1

0 2 4 6 8 10 12 14 16 18 20
Example

SPCP Scheduling
Example

SPCP Scheduling
Example

SPCP Scheduling
Example

SPCP Scheduling
SPCP Scheduling

Example

P5  P4  P3  P2  P1

0  2  4  6  8  10  12  14  16  18  20
Example

SPCP Scheduling

P1

P2

P3

P4

P5
Example

SPCP Scheduling
Comparison with Priority Ceiling Protocol
Analysis: Stack-Based Priority Ceiling

- **Pros**
  - Simple to implement.
  - Slightly better worst-case when compared to normal PCP – two less context switches.
  - No priority inheritance needed.

- **Cons**
  - Threads cannot self suspend.
Summary

- 4 protocols controlling resource access in priority driven preemptive systems
  - NPCs
  - PI
  - PCP
  - SPCP
Summary

- NPCs and PI do not require a priori knowledge of resource requirements
- PI neither prevents deadlocks nor avoids deadlocks
- All protocols -except PI- ensure that processes are blocked \textit{at most once}\footnote{N.}