Operating systems

Module 10

Deadlocks

Part 1
MODULE 10 – DEADLOCKS – PART 1

• Deadlocks in Semaphores
• Priority Inversion concept
• System Model
• Deadlock Characterization
• Methods for Handling Deadlocks
• Deadlock Prevention
**DEADLOCK IN SEMAPHORE**

- **Deadlock** – two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes

- Let $s$ and $q$ be two semaphores initialized to 1

  \[
  \begin{align*}
  P_0 & \quad P_1 \\
  \text{wait}(S) & \quad \text{wait}(Q); \\
  \text{wait}(Q); & \quad \text{wait}(S); \\
  \ldots & \quad \ldots \\
  \text{signal}(S); & \quad \text{signal}(Q); \\
  \text{signal}(Q); & \quad \text{signal}(S);
  \end{align*}
  \]
STARVATION IN SEMAPHORE/PRIORITY INVERSION CONCEPT

• **Starvation** – indefinite blocking
  - A process may never be removed from the semaphore queue in which it is suspended

• **Priority Inversion** – Scheduling problem when lower-priority process holds a lock needed by higher-priority process
  - Solved via *priority-inheritance protocol*
SYSTEM MODEL

• System consists of resources

• Resource types $R_1, R_2, \ldots, R_m$
  
  CPU cycles, memory space, I/O devices

• Each process utilizes a resource as follows:
  
  o request
  o use
  o release
DEADLOCK CHARACTERIZATION

Deadlock can arise if four conditions hold simultaneously:

- Mutual exclusion
- Hold and wait
- No preemption
- Circular wait
METHODS FOR HANDLING DEADLOCKS

- Ensure that the system will *never* enter a deadlock state (Prevent/Avoid)
- Allow the system to enter a deadlock state and then recover (Detect and Recover)
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems
DEADLOCK PREVENTION

Restrain the ways request can be made:

- **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable resources

- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources

- **No Preemption** –
  - Add preemption

- **Circular Wait** –
  impose a total ordering of all resource types
MODUe 10 – DEADLOCKS – PART I

- Deadlocks in Semaphores
- Priority Inversion concept
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
Operating systems

Module 10
Deadlocks – Deadlock Avoidance
Bankers’ algorithm

Part 11
UNIT 10 – DEADLOCKS

- Deadlock Avoidance
- Deadlock Detection
DEADLOCK AVOIDANCE

Requires that the system has some additional \textit{a priori} information available

- Simplest and most useful model requires that each process declare the \textit{maximum number} of resources of each type that it may need

- Dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

- Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes
DEADLOCK AVOIDANCE – SAFE STAE

- Resource Allocation Graph
### BANKERS ALGORITHM

R0 has 8 instances, R1 has 5 instances, R2 has 9 instances, R3 has 8 instances

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R0</strong></td>
<td><strong>R1</strong></td>
<td><strong>R2</strong></td>
</tr>
<tr>
<td>P0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### Available

<table>
<thead>
<tr>
<th><strong>R0</strong></th>
<th><strong>R1</strong></th>
<th><strong>R2</strong></th>
<th><strong>R3</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

Yes it is safe, one sequence is P3, P0, P1, P2, P4
BANKERS ALGORITHM

R0 has 8 instances, R1 has 5 instances, R2 has 9 instances, R3 has 7 instances

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>R0</strong></td>
<td><strong>R1</strong></td>
<td><strong>R2</strong></td>
</tr>
<tr>
<td>P0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td><strong>P0</strong></td>
<td><strong>3</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td><strong>P1</strong></td>
<td><strong>0</strong></td>
<td><strong>2</strong></td>
</tr>
<tr>
<td><strong>P2</strong></td>
<td><strong>5</strong></td>
<td><strong>1</strong></td>
</tr>
<tr>
<td><strong>P3</strong></td>
<td><strong>1</strong></td>
<td><strong>4</strong></td>
</tr>
<tr>
<td><strong>P4</strong></td>
<td><strong>3</strong></td>
<td><strong>0</strong></td>
</tr>
</tbody>
</table>

Available

R0 has 8 instances, R1 has 5 instances, R2 has 9 instances, R3 has 7 instances

No, it is NOT safe
DEADLOCK DETECTION

• Allow system to enter deadlock state
• Detection algorithm
• Recovery scheme