CSC 4103 - Operating Systems  
Fall 2009

Lecture XII  
Deadlocks - III

Tevfik Koşar  
Louisiana State University  
October 6th, 2009

Deadlock Detection

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

Single Instance of Each Resource Type

- Maintain wait-for graph  
- Nodes are processes.  
- \( P_i \rightarrow P_j \) if \( P_i \) is waiting for \( P_j \).

Single Instance of Each Resource Type

- Periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in a graph requires an order of \( n^2 \) operations, where \( n \) is the number of vertices in the graph.
- Only good for single-instance resource allocation systems.

Several Instances of a Resource Type

- \textit{Available}: A vector of length \( m \) indicates the number of available resources of each type.
- \textit{Allocation}: An \( n \times m \) matrix defines the number of resources of each type currently allocated to each process.
- \textit{Request}: An \( n \times m \) matrix indicates the current request of each process. If \( \text{Request} \left[ i, j \right] = k \), then process \( P_i \) is requesting \( k \) more instances of resource type \( R_j \).

Detection Algorithm

1. Let \( \textit{Work} \) and \( \textit{Finish} \) be vectors of length \( m \) and \( n \), respectively. Initialize:
   (a) \( \textit{Work} = \textit{Available} \)
   (b) For \( i = 0, 2, \ldots, n-1 \), if \( \textit{Allocation} \left[ i, \cdot \right] = 0 \), then \( \textit{Finish} \left[ i \right] = \text{false} \); otherwise, \( \textit{Finish} \left[ i \right] = \text{true} \).
2. Find an index \( i \) such that both:
   (a) \( \textit{Finish} \left[ i \right] = \text{false} \)
   (b) \( \textit{Request}_i \leq \textit{Work}_i \)

If no such \( i \) exists, go to step 4.
Detection Algorithm (Cont.)

3. \( \text{Work} = \text{Work} + \text{Allocation}, \)
   \( \text{Finish}[i] = \text{true} \)
   go to step 2.

4. If \( \text{Finish}[i] == \text{false}, \) for some \( i, 0 \leq i \leq n-1, \) then the
   system is in deadlock state. Moreover, if \( \text{Finish}[i] == \text{false} \), then \( P_i \) is deadlocked.

Algorithm requires an order of \( O(m \times n^2) \) operations to detect whether the
system is in deadlocked state.

Example of Detection Algorithm

4. If \( \text{Finish}[i] == \text{false}, \) for some \( i, 0 \leq i \leq n-1, \) then the
   system is in deadlock state. Moreover, if \( \text{Finish}[i] == \text{false} \), then \( P_i \) is deadlocked.

Algorithm requires an order of \( O(m \times n^2) \) operations to detect whether the
system is in deadlocked state.

Example of Detection Algorithm

- Five processes \( P_0 \) through \( P_4; \) three resource types
  - \( A \) (7 instances), \( B \) (2 instances), and \( C \) (6 instances).
- Snapshot at time \( T_0: \)

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Request</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) B C</td>
<td>0 1 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>( P_0 )</td>
<td>2 0 0</td>
<td>2 0 2</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>3 0 3</td>
<td>0 0 0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>2 1 1</td>
<td>1 0 0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>0 0 2</td>
<td>0 0 2</td>
</tr>
</tbody>
</table>
- Sequence \( <P_0, P_2, P_3, P_1, P_4> \) will result in \( \text{Finish}[i] = \text{true} \) for all \( i. \)

Example (Cont.)

- \( P_2 \) requests an additional instance of type \( C. \)

<table>
<thead>
<tr>
<th>Request</th>
<th>Allocation</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A ) B C</td>
<td>0 0 0</td>
<td></td>
</tr>
<tr>
<td>( P_0 )</td>
<td>2 0 1</td>
<td></td>
</tr>
<tr>
<td>( P_1 )</td>
<td>0 0 1</td>
<td></td>
</tr>
<tr>
<td>( P_3 )</td>
<td>1 0 0</td>
<td></td>
</tr>
<tr>
<td>( P_4 )</td>
<td>0 0 2</td>
<td></td>
</tr>
</tbody>
</table>
- State of system?
  - Can reclaim resources held by process \( P_0, \) but insufficient resources to fulfill other processes; requests.
  - Deadlock exists, consisting of processes \( P_1, P_2, P_3, \) and \( P_4. \)

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim - minimize cost.
- Rollback - return to some safe state, restart process for that state.
- Starvation - same process may always be picked as victim, include number of rollback in cost factor.

Deadlock Avoidance

Requirements that the system has some additional \( a \) priori information available.

- Simplest and most useful model requires that each process declare the maximum number of resources of each type that it may need.
- The deadlock-avoidance algorithm 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.
**Safe State**

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

- System is in safe state if there exists a safe sequence of all processes.

- Sequence \(<P_1, P_2, ..., P_n>\) is safe if for each \(P_i\), the resources that \(P_i\) can still request can be satisfied by currently available resources + resources held by all the \(P_j\), with \(j<i\).
  - If \(P_i\) resource needs are not immediately available, then \(P_i\) can wait until all \(P_j\) have finished.
  - When \(P_i\) is finished, \(P_i\) can obtain needed resources, execute, return allocated resources, and terminate.
  - When \(P_i\) terminates, \(P_{i+1}\) can obtain its needed resources, and so on.

**Basic Facts**

- If a system is in safe state ⇒ no deadlocks.
- If a system is in unsafe state ⇒ possibility of deadlock.
- Avoidance ⇒ ensure that a system will never enter an unsafe state.

**Resource-Allocation Graph Algorithm**

- Claim edge \(P_i \rightarrow R_j\) indicated that process \(P_i\) may request resource \(R_j\); represented by a dashed line.

- Claim edge converts to request edge when a process requests a resource.

- When a resource is released by a process, assignment edge reconverts to a claim edge.

- Resources must be claimed \(a\ priori\) in the system.

**Unsafe State In Resource-Allocation Graph**

**Resource-Allocation Graph For Deadlock Avoidance**
Banker’s Algorithm

- Multiple instances.
- Each process must a priori claim maximum use.
- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.

Data Structures for the Banker’s Algorithm

Let \( n \) = number of processes, and \( m \) = number of resources types.

- **Available**: Vector of length \( m \). If available \([j] = k\), there are \( k \) instances of resource type \( R_j \) available.
- **Max**: \( n \times m \) matrix. If \( \text{Max}[i,j] = k \), then process \( P_i \) may request at most \( k \) instances of resource type \( R_j \).
- **Allocation**: \( n \times m \) matrix. If \( \text{Allocation}[i,j] = k \) then \( P_i \) is currently allocated \( k \) instances of \( R_j \).
- **Need**: \( n \times m \) matrix. If \( \text{Need}[i,j] = k \), then \( P_i \) may need \( k \) more instances of \( R_j \) to complete its task.

\[ \text{Need}[i,j] = \text{Max}[i,j] - \text{Allocation}[i,j]. \]

Safety Algorithm

1. Let \( \text{Work} \) and \( \text{Finish} \) be vectors of length \( m \) and \( n \), respectively. Initialize:

\[ \text{Work} = \text{Available} \]

\[ \text{Finish}[i] = \text{false} \text{ for } i = 1, 3, \ldots, n. \]

2. Find an \( i \) such that both:
   (a) \( \text{Finish}[i] = \text{false} \)
   (b) \( \text{Need}_i \leq \text{Work} \)

   If no such \( i \) exists, go to step 4.

3. \( \text{Work} = \text{Work} + \text{Allocation}_i \)
   \( \text{Finish}[i] = \text{true} \)

   go to step 2.

4. If \( \text{Finish}[i] = \text{true} \text{ for all } i \), then the system is in a safe state.

Resource-Request Algorithm for Process \( P_i \)

\( \text{Request} = \text{request vector for process } P_i \). If
\( \text{Request}[j] = k \) then process \( P_i \) wants \( k \) instances of resource type \( R_j \).

1. If \( \text{Request}_i \leq \text{Need}_i \) go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.

2. If \( \text{Request}_i \leq \text{Available}_i \), go to step 3. Otherwise \( P_i \) must wait, since resources are not available.

3. Pretend to allocate requested resources to \( P_i \) by modifying the state as follows:

\[ \text{Available} = \text{Available} - \text{Request}; \]
\[ \text{Allocation} = \text{Allocation} + \text{Request}; \]
\[ \text{Need} = \text{Need} - \text{Request}. \]

- If \( \text{safe} \rightarrow \) the resources are allocated to \( P_i \).
- If \( \text{unsafe} \rightarrow P_i \) must wait, and the old resource-allocation state is restored.

Example of Banker’s Algorithm

- 5 processes \( P_0 \) through \( P_4 \); 3 resource types \( A \) (10 instances), \( B \) (5 instances), and \( C \) (7 instances).
- Snapshot at time \( T_0 \):

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Max</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A )</td>
<td>( B )</td>
<td>( C )</td>
</tr>
<tr>
<td>( P_0 )</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>( P_1 )</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>( P_2 )</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>( P_3 )</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>( P_4 )</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Example (Cont.)

- The content of the matrix. Need is defined to be \( \text{Max} - \text{Allocation} \).

| \( \text{Need} \) |
| \( A \) | \( B \) | \( C \) |
| \( P_0 \) | 7 | 4 | 3 |
| \( P_1 \) | 1 | 2 | 2 |
| \( P_2 \) | 6 | 0 | 0 |
| \( P_3 \) | 0 | 1 | 1 |
| \( P_4 \) | 4 | 3 | 1 |
Example \( P_1 \) Request \((1,0,2)\) (Cont.)

- Check that Request \( \preceq \) Available (that is, \((1,0,2) \preceq (3,3,2) \Rightarrow \text{true})

<table>
<thead>
<tr>
<th>Allocation</th>
<th>Need</th>
<th>Available</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_0)</td>
<td>0 1 0</td>
<td>7 4 3 2 3 0</td>
</tr>
<tr>
<td>(P_1)</td>
<td>3 0 2</td>
<td>0 2 0</td>
</tr>
<tr>
<td>(P_2)</td>
<td>3 0 1</td>
<td>6 0 0</td>
</tr>
<tr>
<td>(P_3)</td>
<td>2 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>(P_4)</td>
<td>0 0 2</td>
<td>4 3 1</td>
</tr>
</tbody>
</table>

- Executing safety algorithm shows that sequence \(<P_1, P_3, P_4, P_0, P_2>\) satisfies safety requirement.
- Can request for \((3,3,0)\) by \(P_4\) be granted?
- Can request for \((0,2,0)\) by \(P_0\) be granted?

Acknowledgements

- “Modern Operating Systems” book and supplementary material by A. Tanenbaum
- R. Doursat and M. Yuksel from UNR