CS370 Operating Systems
Colorado State University
Yashwant K Malaiya
Fall 2017  Lecture 17

Deadlock

Slides based on
• Text by Silberschatz, Galvin, Gagne
• Various sources
• Claim edge may request request edge – assignment edge?
• How to avoid cycles? By design/allocation
• How to find the safe sequences? Exhaustive search?
• Why is it OK to ignore deadlocks? You mean possibility of.
Methods for Handling Deadlocks

• Ensure that the system will never enter a deadlock state:
  – Deadlock prevention
    • ensuring that at least one of the 4 conditions cannot hold
  – Deadlock avoidance
    • Dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

• Allow the system to enter a deadlock state
  – Detect and then recover. Hope is that it happens rarely.

• Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.
Banker’s Algorithm: examining a request

- Multiple instances of resources.
- Each process must a priori claim maximum use
- When a process requests a resource
  - it may have to wait (resource request algorithm)
  - Request not granted if the resulting system state is unsafe (safety algorithm)
- When a process gets all its resources it must return them in a finite amount of time
- Modeled after a banker in a small town making loans
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.

**Processes vs resources:**

- **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: Is System in safe state?

1. Let Work and Finish be vectors of length m and n, respectively. Initialize:
   - Work = Initially Available resources
   - Finish[i] = false for i = 0, 1, ..., n-1

2. Find a process i such that both:
   (a) Finish[i] = false
   (b) Need_i ≤ Work
   If no such i exists, go to step 4

3. Work = Work + Allocation_i
   Finish[i] = true
   go to step 2

4. If Finish[i] == true for all i, then the system is in a safe state

n = number of processes,
m = number of resources types
Need_i: additional res needed
Work: res currently free
Finish_i: processes finished
Allocation_i: allocated to i
Resource-Request Algorithm for Process $P_i$

Notation: $Request_i = $ request vector for process $P_i$. If $Request_i[j] = k$ then process $P_i$ wants $k$ instances of resource type $R_j$.

Algorithm: Should the allocation request be granted?

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim.
2. If $Request_i \leq Available$, go to step 3. Otherwise $P_i$ must wait, since resources are not available.
3. Is allocation safe?: Pretend to allocate requested resources to $P_i$ by modifying the state as follows:
   
   $$
   \begin{align*}
   Available &= Available - Request_i; \\
   Allocation_i &= Allocation_i + Request_i; \\
   Need_i &= Need_i - Request_i;
   \end{align*}
   $$

   - If safe $\Rightarrow$ the resources are allocated to $P_i$
   - If unsafe $\Rightarrow P_i$ must wait, and the old resource-allocation state is preserved.
Example A: Banker’s Algorithm

- 5 processes $P_0$ through $P_4$;
- 3 resource types: $A$ (10 instances), $B$ (5 instances), and $C$ (7 instances)
- Is it a safe state?

<table>
<thead>
<tr>
<th>Process</th>
<th>Max A</th>
<th>Max B</th>
<th>Max C</th>
<th>Allocation A</th>
<th>Allocation B</th>
<th>Allocation C</th>
<th>Need A</th>
<th>Need B</th>
<th>Need C</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>available</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>
Example A: Banker’s Algorithm

- Is it a safe state?
- Yes, since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Allocation</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>available</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

P1 run to completion. Available becomes \([3 \ 3 \ 2]+[2 \ 0 \ 0] = [5 \ 3 \ 2]\)
P3 run to completion. Available becomes \([5 \ 3 \ 2]+[2 \ 1 \ 1] = [7 \ 4 \ 3]\)
P4 run to completion. Available becomes \([7 \ 4 \ 3]+[0 \ 0 \ 2] = [7 \ 4 \ 5]\)
P2 run to completion. Available becomes \([7 \ 4 \ 5]+[3 \ 0 \ 2] = [10 \ 4 \ 7]\)
P0 run to completion. Available becomes \([10 \ 4 \ 7]+[0 \ 1 \ 0] = [10 \ 5 \ 7]\)

Hence state above is safe.
Ex B: Assume now $P_1$ Requests (1,0,2)

- Check that Request ≤ Available. $(1,0,2) \leq (3,3,2) \rightarrow true.$
- Check for safety after pretend allocation. $P_1$ allocation would be $(2 0 0) + (1 0 2) = 302$

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Allocation</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>available</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

Sequence $< P_1, P_3, P_4, P_0, P_2 >$ satisfies safety requirement.

Hence state above is safe, thus the allocation would be safe.
Ex C,D: Additional Requests ..

- Given State is (same as previous slide)

<table>
<thead>
<tr>
<th>Process</th>
<th>Max</th>
<th>Allocation</th>
<th>Need</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>available</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>

P4 request for (3,3,0): cannot be granted - resources are not available.
P0 request for (0,2,0): cannot be granted since the resulting state is unsafe.
Bankers Algorithm: Practical Issues

• Processes rarely know in advance about their maximum resource needs
• Number of processes is not fixed
  – Varies dynamically
• Resources thought to be available can disappear
• Few systems use this algorithm
Deadlock Detection

• Allow system to enter deadlock state
• Detection algorithm
  – Single instance of each resource:
    • wait-for graph
  – Multiple instances:
    • detection algorithm (based on Banker’s algorithm)
• Recovery scheme
Single Instance of Each Resource Type

- Maintain **wait-for graph** (based on resource allocation graph)
  - Nodes are processes
  - $P_i \rightarrow P_j$ if $P_i$ is waiting for $P_j$
  - *Deadlock if cycles*

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

- 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
Resource-Allocation Graph and Wait-for Graph

Resource-Allocation Graph

Corresponding wait-for graph

Has cycles. Deadlock.
Banker’s algorithm: Can requests by all processes be satisfied?

• **Available:** A vector of length $m$ indicates the number of available (currently free) resources of each type.

• **Allocation:** An $n \times m$ matrix defines the number of resources of each type currently allocated to each process.

• **Request:** An $n \times m$ matrix indicates the current request of each process. If $Request[i][j] = k$, then process $P_i$ is requesting $k$ more instances of resource type $R_j$. 
Detection Algorithm

1. Let $Work$ and $Finish$ be vectors of length $m$ and $n$, respectively. Initialize:
   (a) $Work = Available$
   (b) For $i = 1, 2, ..., n$, if $Allocation_i \neq 0$, then $Finish[i] = false$; otherwise, $Finish[i] = true$

2. Find an index $i$ such that both:
   (a) $Finish[i] == false$
   (b) $Request_i \leq Work$

   If no such $i$ exists, go to step 4

3. $Work = Work + Allocation_i$
   $Finish[i] = true$

   go to step 2 (find next process)

4. If $Finish[i] == false$, for some $i$, $1 \leq i \leq n$, then the system is in deadlock state. Moreover, if $Finish[i] == false$, 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

$n = \text{number of processes,}$
$m = \text{number of resources types}$
$\text{Need}_i: \text{additional res needed}$
$\text{Work}: \text{res currently free}$
$\text{Finish}_i: \text{processes finished}$
$\text{Allocation}_i: \text{allocated to i}$
Example of Detection Algorithm

- Five processes $P_0$ through $P_4$; three resource types A (7 instances), B (2 instances), and C (6 instances)
- Sequence $<P_0, P_2, P_3, P_1, P_4>$ will result in $Finish[i] = true$ for all $i$. **No deadlock**

<table>
<thead>
<tr>
<th>Process</th>
<th>Allocation</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A  B  C</td>
<td>A  B  C</td>
</tr>
<tr>
<td>available</td>
<td>0  0  0</td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>0  1  0</td>
<td>0  0  0</td>
</tr>
<tr>
<td>P1</td>
<td>2  0  0</td>
<td>2  0  2</td>
</tr>
<tr>
<td>P2</td>
<td>3  0  3</td>
<td>0  0  0</td>
</tr>
<tr>
<td>P3</td>
<td>2  1  1</td>
<td>1  0  0</td>
</tr>
<tr>
<td>P4</td>
<td>0  0  2</td>
<td>0  0  2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>After</th>
<th>available</th>
</tr>
</thead>
<tbody>
<tr>
<td>ini</td>
<td>0  0  0</td>
</tr>
<tr>
<td>P0</td>
<td>0  1  0</td>
</tr>
<tr>
<td>P2</td>
<td>3  1  3</td>
</tr>
<tr>
<td>P3</td>
<td>5  2  4</td>
</tr>
<tr>
<td>P1</td>
<td>7  2  4</td>
</tr>
<tr>
<td>P4</td>
<td>7  2  6</td>
</tr>
</tbody>
</table>
Example of Detection Algorithm (cont)

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

<table>
<thead>
<tr>
<th>Process</th>
<th>Allocation</th>
<th>Request</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>A  B  C</td>
<td>A  B  C</td>
</tr>
<tr>
<td>available</td>
<td>0  0  0</td>
<td></td>
</tr>
<tr>
<td>P0</td>
<td>0  1  0</td>
<td>0  0  0</td>
</tr>
<tr>
<td>P1</td>
<td>2  0  0</td>
<td>2  0  2</td>
</tr>
<tr>
<td>P2</td>
<td>3  0  3</td>
<td>0  0  1</td>
</tr>
<tr>
<td>P3</td>
<td>2  1  1</td>
<td>1  0  0</td>
</tr>
<tr>
<td>P4</td>
<td>0  0  2</td>
<td>0  0  2</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 \( P_4 \)
• iClicker Quiz
Detection-Algorithm Usage

• When, and how often, to invoke depends on:
  – How often a deadlock is likely to occur
  – How many processes will need to be rolled back
    • one for each disjoint cycle

• If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.
Recovery from Deadlock: Process Termination

Choices
- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated

In which order should we choose to abort?
1. Priority of the process
2. How long process has computed, and how much longer to completion
3. Resources the process has used
4. Resources process needs to complete
5. How many processes will need to be terminated
6. Is process interactive or batch?
• **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 recovery through rollbacks

- **Checkpoint** process periodically
  - Contains memory image and resource state
- Deadlock detection tells us which resources are needed
- Process owning a needed resource
  - **Rolled back** to before it acquired needed resource
    - Work done since rolled back checkpoint discarded
  - **Assign** resource to deadlocked process
In a livelock two processes need each other’s resource

- Both run and make no progress, but neither process blocks
- Use CPU quantum over and over without making progress

Ex: If fork fails because process table is full

- Wait for some time and try again
- But there could be a collection of processes each trying to do the same thing
- Avoided by ensuring that only one process (chosen randomly or by priority) takes action