# Chapter 7: Deadlock Detection and Recovery

CSCI 3753 Operating Systems

Instructor: Chris Womack

University of Colorado at Boulder

All material by Dr. Rick Han

# Recap: Banker's Algorithm for Deadlock Avoidance

- Is the system in a safe state? Find a safe sequence:
  1. Let Work and Finish be vectors length m and n respectively. Initialize Work = Available, and Finish[i]=false for i=0,...,n-1
  2. Find a process i such that both
     - Finish[i]==false, and
     - $Need_i \leq Work$

     If no such i exists, go to step 4.
  3. $Work = Work + Alloc_i$

     Finish[i] = true

     Go to step 2.
  4. If Finish[i]==true for all i, then the system is in a safe state

Intuition: if all prior processes give up all their resources, is there enough to meet the max needs of next process in the sequence?

# Recap: Resource-Request Algorithm

Let $Request_i$ be a new request vector for resources for process $P_i$

1. If $Request_i \le Need_i$ , go to step 2. Else, the new request exceeds the maximum claim. Exit.
2. If $Request_i \le Available$, go to step 3. Else, process $P_i$ must wait because there aren't enough available resources
3. Temporarily modify Available[j], Need[i,j], and Alloc[i,j]
   - Avail -= $Request_i$
   - $Alloc_i$ += $Request_i$
   - $Need_i$ -= $Request_i$

   Execute the Banker's algorithm. If system is in a safe state, grant the request and update Avail, Need, and Alloc.

# Recap: Deadlock Detection

Assume we have a matrix **Alloc[I,j]** of m resources allocated to n processes. Assume also we have a matrix **Request[i,j]** of m resources already requested by the m processes. (no more Max or Need)

1. Let Work and Finish be vectors length m and n respectively. Initialize Work = Available. For i=0,…n-1, if $Alloc_i$==0 then Finish[i]=true, else Finish[i]=false

No hold and wait so no deadlock

2. Find a process i such that both
   - Finish[i]==false, and
   - $Request_i \leq Work$

   If no such i exists, go to step 4.
3. Work = Work + $Alloc_i$

   Finish[i] = true

   Go to step 2.

Intuition: if all prior processes release all their resources, is there enough to meet the requested needs of remaining processes?

4. If Finish[i]==false for some i, then the system is in a deadlocked state. Moreover, if Finish[i]==false, then process $P_i$ is deadlocked

# Deadlock Detection

- Example 1:
  - 3 resources (A,B,C) with total instances equal to (7,2,6)
  - 5 processes
  - At time t0, the allocated resources Alloc[i,j], requested resources Request[i,j], and Available resources Avail[j], are:

$$
\text{Alloc}[i,j] = \begin{array}{c} \\ P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{array} \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \left[\begin{array}{ccc} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{array}\right] \end{array}
\quad
\text{Request}[i,j] = \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \left[\begin{array}{ccc} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{array}\right] \end{array}
\quad
\text{Avail}[j] = \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \left[\begin{array}{ccc} 0 & 0 & 0 \end{array}\right] \end{array}
$$

Is this system deadlocked? No, the detection algorithm finds a sequence <P0,P2,P3,P1,P4> that releases all resources such that Finish[i]==true for all i.

# Deadlock Detection

Alloc[i,j]

$$\begin{array}{c|ccc} & A & B & C \\ \hline P0 & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ P1 & 2 & 0 & 0 \\ P2 & 3 & 0 & 3 \\ P3 & 2 & 1 & 1 \\ P4 & 0 & 0 & 2 \end{array}$$

Request[i,j]

$$\begin{array}{c|ccc} & A & B & C \\ \hline P0 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ P1 & 2 & 0 & 2 \\ P2 & 0 & 0 & 0 \\ P3 & 1 & 0 & 0 \\ P4 & 0 & 0 & 2 \end{array}$$

Avail[j]

$$\begin{array}{ccc} A & B & C \\ \hline 0 & 0 & 0 \end{array}$$

- Find a process *i* such that $Request_i \leq$ Work (= Available = <0,0,0>)
  - P0's $Request_0$ = <0,0,0> ≤ Work = <0,0,0>
  - Work = Work + $Alloc_0$ = <0,0,0> + <0,1,0> = <0,1,0>

# Deadlock Detection

Alloc[i,j] Request[i,j]

$$
\begin{array}{c}
 \\ P0 \\ P1 \\ P2 \\ P3 \\ P4
\end{array}
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\left[\begin{array}{ccc} 0 & 1 & 0 \\ 2 & 0 & 0 \\ \mathbf{3} & \mathbf{0} & \mathbf{3} \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{array}\right]
\end{array}
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\left[\begin{array}{ccc} 0 & 0 & 0 \\ 2 & 0 & 2 \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{array}\right]
\end{array}
$$

Avail[j]

$$
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\left[\begin{array}{ccc} 0 & 0 & 0 \end{array}\right]
\end{array}
$$

- Find a process *i* such that $\text{Request}_i \leq$ Work = <0,1,0>
  - No process except P2 satisfies this inequality
  - P2's $\text{Request}_2$ = <0,0,0> ≤ Work = <0,1,0>
  - Work = Work + $\text{Alloc}_2$ = <0,1,0> + <3,0,3> = <3,1,3>

# Deadlock Detection

Alloc[i,j]

$$
\begin{array}{c}
 \\ P0 \\ P1 \\ P2 \\ P3 \\ P4
\end{array}
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}
\end{array}
$$

Request[i,j]

$$
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}
\end{array}
$$

Avail[j]

$$
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

- Find a process *i* such that $\text{Request}_i \leq$ Work = <3,1,3>
  - All 3 remaining processes P1, P3 and P4 satisfy this inequality, i.e. their requests can be met by what is available in Work
  - Choose them in any order, say P3, P1, P4
  - Thus, <P0, P2, P3, P1, P4> satisfies all requests without deadlock

University of Colorado **Boulder**

# Deadlock Detection

- Example 2:
  - Same as Example 1, except P2 requests one more instance of C
- Is this system deadlocked?
  - Yes, the detection algorithm finds P1, P2, P3 and P4 are all deadlocked.

$$
\begin{array}{c} \\ P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{array}
\begin{array}{c} \text{Alloc[i,j]} \\ \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix} \end{array}
\quad
\begin{array}{c} \text{Request[i,j]} \\ \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & \color{red}{1} \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix} \end{array}
\quad
\begin{array}{c} \text{Avail[j]} \\ \begin{array}{ccc} A & B & C \end{array} \\ \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \end{array}
$$

# Deadlock Detection

- Find a process *i* such that $Request_i \leq Work$ (= Available = $<0,0,0>$)
  - P0's $Request_0 = <0,0,0> \leq Work = <0,0,0>$
  - $Work = Work + Alloc_0 = <0,0,0> + <0,1,0> = <0,1,0>$

$$
\begin{array}{cc}
 & \text{Alloc[i,j]} \\
 & \begin{array}{ccc} A & B & C \end{array} \\
\begin{array}{c} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{array} &
\begin{bmatrix} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
\text{Request[i,j]} \\
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 0 & 0 \\ 2 & 0 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \end{bmatrix}
\end{array}
\quad
\begin{array}{c}
\text{Avail[j]} \\
\begin{array}{ccc} A & B & C \end{array} \\
\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}
\end{array}
$$

# Deadlock Detection

- Find a process *i* such that $\text{Request}_i \leq \text{Work} = <0,1,0>$
  - No process satisfies this inequality
  - P1's $\text{Request}_1 = <2,0,2> \not\leq <0,1,0>$. Same for P2, P3, P4
  - Therefore, P1, P2, P3 and P4 are in deadlock. They wait forever, because their requests will never be fulfilled.

Alloc[i,j] Request[i,j]

$$
\begin{array}{c c}
 & \begin{array}{ccc} A & B & C \end{array} \\
\begin{array}{c} P0 \\ P1 \\ P2 \\ P3 \\ P4 \end{array} &
\left[\begin{array}{ccc} 0 & 1 & 0 \\ 2 & 0 & 0 \\ 3 & 0 & 3 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{array}\right]
\end{array}
\quad
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\left[\begin{array}{ccc} 0 & 0 & 0 \\ \mathbf{2} & \mathbf{0} & \mathbf{2} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{2} \end{array}\right]
\end{array}
$$

Avail[j]

$$
\begin{array}{c}
\begin{array}{ccc} A & B & C \end{array} \\
\left[\begin{array}{ccc} 0 & 0 & 0 \end{array}\right]
\end{array}
$$

# Deadlock Detection

- When/how often should the detection algorithm run?
  - Depends on how often deadlock is likely to occur
  - Depends on how quickly deadlock grows after it occurs, i.e. how many processes get pulled into deadlock and on what time scale

1. Could check at each resource request – this is costly
2. Could check periodically – but what is a good time interval?
3. Could check if CPU utilization suddenly drops
   - this might be an indication that there's deadlock, and processes are no longer executing, but what's a good threshold?
4. Could check if resource utilization exceeds some threshold, but what's a good threshold?

# Deadlock Recovery

- After OS has detected which processes are deadlocked, then OS can:
  1. Terminate all processes - draconian
  2. Terminate one process at a time until the deadlock cycle is eliminated
     - Check if there is still deadlock after each process is terminated. If not, then stop.
  3. Preempt some processes – temporarily take away a resource from current owner and give it to another process but don't terminate process
     - e.g. give access to a laser printer – this is risky if you're in middle or printing a document
  4. Rollback some processes to a checkpoint – assuming that processes have saved their state at some checkpoint