# 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:
  - 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<sub>i</sub> ≤ Work
       If no such i exists, go to step 4.
  - 3. Work = Work + Alloc<sub>i</sub> ← Finish[i] = trueGo to step 2.
- Intuition: if all prior processes give up all their resources, is there enough to meet the max needs of next process in the sequence?
- 4. If Finish[i]==true for all i, then the system is in a safe state



## Recap: Resource-Request Algorithm

Let Request<sub>i</sub> be a new request vector for resources for process P<sub>i</sub>

- If Request<sub>i</sub> ≤ Need<sub>i</sub>, go to step 2. Else, the new request exceeds the maximum claim. Exit.
- If Request<sub>i</sub> ≤ Available, go to step 3. Else, process P<sub>i</sub> must wait because there aren't enough available resources
- Temporarily modify Available[j], Need[i,j], and Alloc[i,j]
  - Avail -= Request<sub>i</sub>
  - Alloc<sub>i</sub> += Request<sub>i</sub>
  - Need<sub>i</sub> -= Request<sub>i</sub>

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)

- Let Work and Finish be vectors length m and n respectively. Initialize Work = Available. For i=0,...n-1, if Alloc<sub>i</sub>==0 then Finish[i]=true, else Finish[i]=false
   Find a process i gueb that both
- 2. Find a process i such that both
  - Finish[i]==false, and
  - Request<sub>i</sub> ≤ Work
     If no such i exists, go to step 4.
- 3. Work = Work + Alloc<sub>i</sub> ← 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?

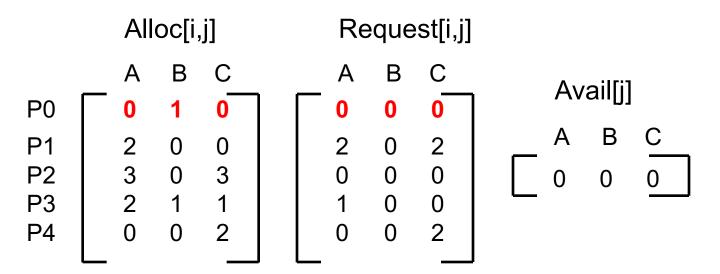
deadlock

4. If Finish[i]==false for some i, then the system is in a deadlocked state. Moreover, if Finish[i]==false, then process P<sub>i</sub> is deadlocked

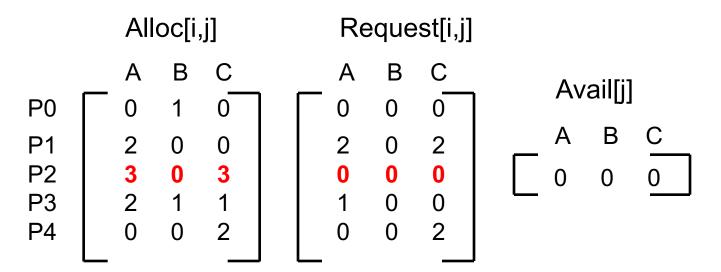
- Example 1:
  - 3 resources (A,B,C) with total instances equal to (7,2,6)
  - 5 processes
  - At time t0, the allocated resources Alloc[i,j], requested resources Request[i,j], and Available resources Avail[j], are:

	All	oc[i,	,j]		Re	que	st[i,j]						
	Α	В	С		Α	В	С			Δ.,	ail[j]		
P0	0	1	0		0	0	0			<b>Λ</b> ν	anı		
P1	2	0	0		2	0	2		_	Α	В	C	_
P2	3	0	3		0	0	0			0	0	0	
P3	2	1	1		1	0	0		_				ر
P4	0	0	2		0	0	2						
								l					

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



- Find a process i such that Request<sub>i</sub> ≤ Work (= Available = <0,0,0>)
  - P0's Request<sub>0</sub> =  $<0,0,0> \le Work = <0,0,0>$
  - Work = Work + Alloc<sub>0</sub> = <0,0,0> + <0,1,0> = <0,1,0>



- Find a process i such that Request<sub>i</sub> ≤ Work = <0,1,0>
  - No process except P2 satisfies this inequality
  - P2's Request<sub>2</sub> =  $<0,0,0> \le Work = <0,1,0>$
  - Work = Work + Alloc<sub>2</sub> = <0,1,0> + <3,0,3> = <3,1,3>

	Alloc[i,j]						Re	que	st[i,j]	]				
		Α	В	С			Α	В	С			Δ.,	ail[j]	
P0		0	1	0			0	0	0	]		<b>~</b> ∨	ануј	
P1		2	0	0			2	0	2			Α	В	<u>C</u>
P2		3	0	3			0	0	0			0	0	0
P3		2	1	1			1	0	0		<u> </u>			
P4		0	0	2			0	0	2					
					J					J				

- Find a process i such that Request<sub>i</sub> ≤ 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



- 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.

	All	oc[i,	j]			Re	que	st[i,j]					
	Α	В	С			Α	В	С			۸.,	<b>~</b> :1Γ:1	
P0	0	1	0	]	Г	0	0	0			AV	ail[j]	
P1	2	0	0			2	0	2		_	Α	В	<u>C</u>
P2	3	0	3			0	0	1			0	0	0
P3	2	1	1			1	0	0		<b>—</b>			
P4	0	0	2			0	0	2					
				J	$ldsymbol{ld}}}}}}$				J				

- Find a process i such that Request<sub>i</sub> ≤ Work (= Available = <0,0,0>)
  - P0's Request<sub>0</sub> =  $<0,0,0> \le Work = <0,0,0>$
  - Work = Work + Alloc<sub>0</sub> = <0,0,0> + <0,1,0> = <0,1,0>

	Alloc[i,j]					Re	que	st[i,j]					
	Α	В	С			Α	В	С			Δν	ail[j]	
P0	0	1	0	]		0	0	0			~v	anı	l
P1	2	0	0			2	0	2			Α	В	<u>C</u>
P2	3	0	3			0	0	1			0	0	0
P3	2	1	1			1	0	0		<u> </u>			
P4	0	0	2			0	0	2					
									J				

- Find a process i such that Request<sub>i</sub> ≤ Work = <0,1,0>
  - No process satisfies this inequality
  - P1's Request<sub>1</sub> = <2,0,2>  $\neq$  <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]					Re	que	st[i,j]					
	Α	В	С	_		Α	В	С	_		۸۰	ail[i]	
P0	0	1	0	1		0	0	0			۸v	ail[j]	
P1	2	0	0			2	0	2		_	Α	В	<u>C</u>
P2	3	0	3			0	0	1			0	0	0
P3	2	1	1			1	0	0		_			
P4	0	0	2			0	0	2					
				J					J				

- 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.
  - 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