CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 26

Deadlock

Slides based on

- **Text by Silberschatz, Galvin, Gagne**
- **Various sources**

FAQ

Relation among: Resource allocation, Safe State and Banker's algorithm

- **Safe State:** If the system can allocate resources to each process in some order, up to the maximum for the process, and still avoid deadlock
- **Banker's algorithm**: When a process requests a resource, it may have to wait (**resource request algorithm**), and request not granted if the resulting system state is unsafe (**safety algorithm**)
- Need [i,j] = Max[i,j] – 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 = Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

2. Find a process ***i*** such that both:

   (a) ***Finish* [*i*] = *false***

   (b) $\boldsymbol{Need_i \leq Work}$

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

3. $\boldsymbol{Work = Work + Allocation_i}$

   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

$\mathbf{Need_i}$: **additional** res needed

Work: res currently free

$\mathbf{Finish_i}$: processes finished

$\mathbf{Allocation_i}$: 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:

$$Available = Available - Request_i;$$
$$Allocation_i = Allocation_i + Request_i;$$
$$Need_i = Need_i - Request_i;$$

Safety Algorithm

- 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.

Colorado State University

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)
- Snapshot at time T_0:

	Allocation	Max	Available
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

- Is it a safe state?

Example (Cont.)

- The matrix ***Need*** is ***Max – Allocation***

	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

Available
A B C
3 3 2

- Next we show that he system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

Example Cont.

The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria, since:

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	3 3 2
P_1	2 0 0	1 2 2	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

P1 can run since need ≤ available

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 $\leq$ Available
  - (1,0,2) $\leq$ (3,3,2) $\Rightarrow$ true. Check for safety after pretend allocation.   P1 allocation would be (2 0 0) + (1 0 2)= 302

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

- Executing safety algorithm shows that sequence < $\boldsymbol{P_1, P_3, P_4, P_0, P_2}$> satisfies safety requirement. Yes, safe state.

Colorado State University

Ex C,D: Additional Requests ..

- Given State is (previous slide)

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	2 3 0
P_1	3 0 2	0 2 0	
P_2	3 0 2	6 0 0	
P_3	2 1 1	0 1 1	
P_4	0 0 2	4 3 1	

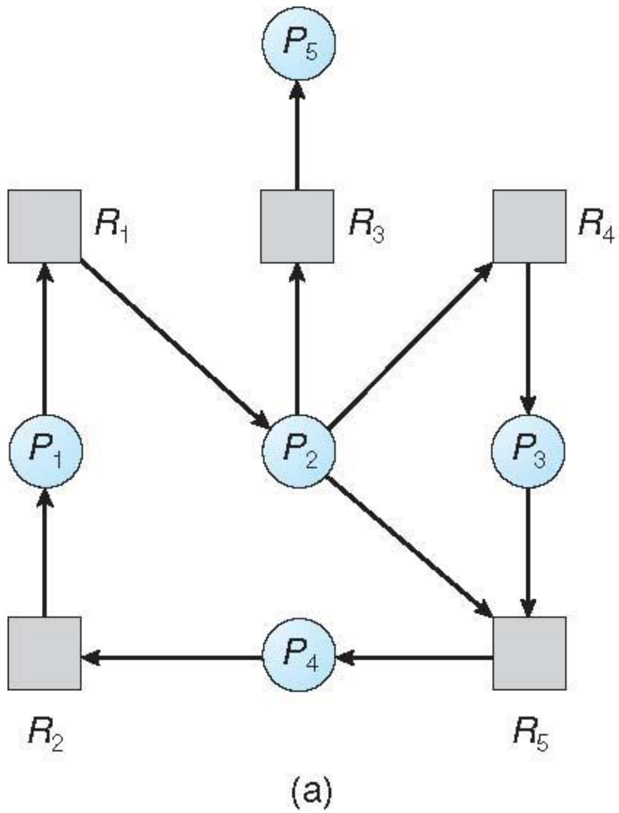
- P4 request for (3,3,0): cannot be granted - resources are not available.
- P0 request for (0,2,0): cannot be granted since the resulting state is unsafe.

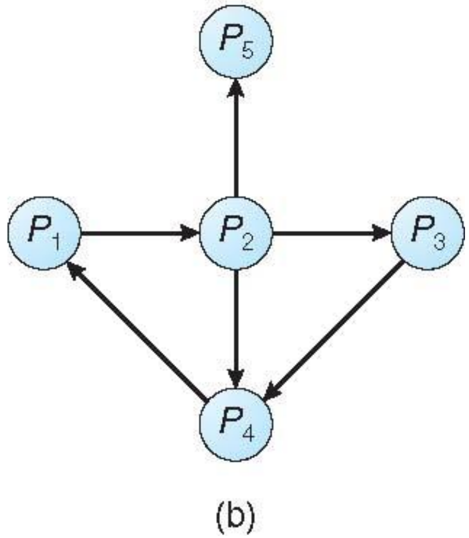
Deadlock Detection

- 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

Corresponding wait-for graph

3 cycles. Deadlock.

Several Instances of a Resource Type

Banker's algorithm: Can requests by all process be satisfied?

- **Available***:* A vector of length ***m*** indicates the number of available (currently free) resources of each type
- **Allocation***:* An ***n*** **x** ***m*** matrix defines the number of resources of each type currently allocated to each process
- **Request***:* An ***n*** **x** ***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 $\boldsymbol{Allocation_i \neq 0}$, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

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

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

Detection Algorithm (Cont.)

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* x *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:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in ***Finish[i] = true*** for all i. ***No deadlock***

Colorado State University

Example (Cont.)

- P_2 requests an additional instance of type C

	Request
	A B C
P_0	0 0 0
P_1	2 0 2
P_2	0 0 1
P_3	1 0 0
P_4	0 0 2

Available		
A	B	C
0	0	0

- 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

Quiz

- 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?

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 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