CSC 4103 - Operating Systems
Spring 2007

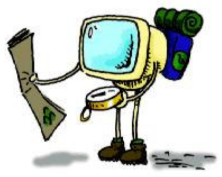
Lecture - IX
Deadlocks - II

Tevfik Koşar

Louisiana State University
February 15th, 2007

Roadmap

- Deadlocks
  - Deadlock Avoidance
  - Deadlock Detection
  - Recovery from Deadlock



Deadlock Avoidance

Requires 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.

3

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_j 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.

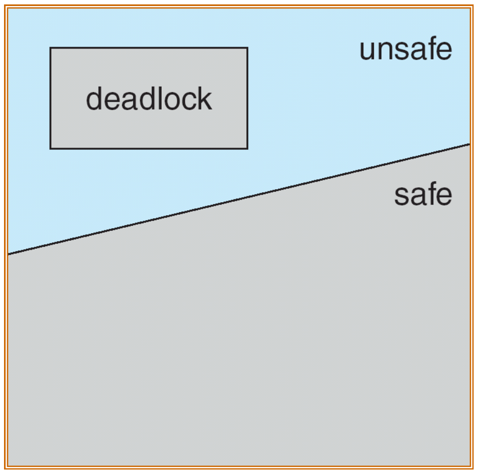
4

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.

5

Safe, Unsafe , Deadlock State



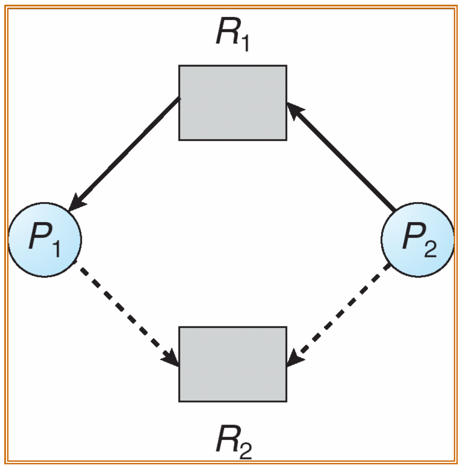
6

Resource-Allocation Graph Algorithm

- *Claim edge* $P_i \rightarrow R_j$ indicated that process P_j 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.

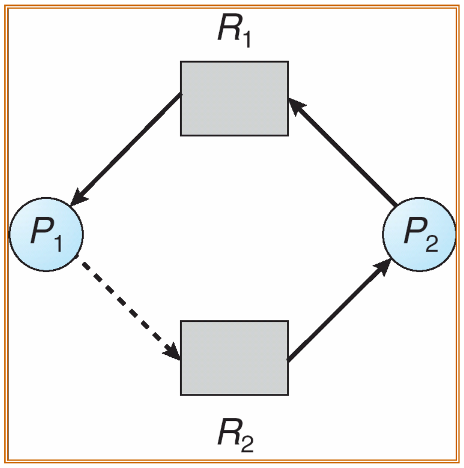
7

Resource-Allocation Graph For Deadlock Avoidance



8

Unsafe State In Resource-Allocation Graph



9

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.

10

Data Structures for the Banker's Algorithm

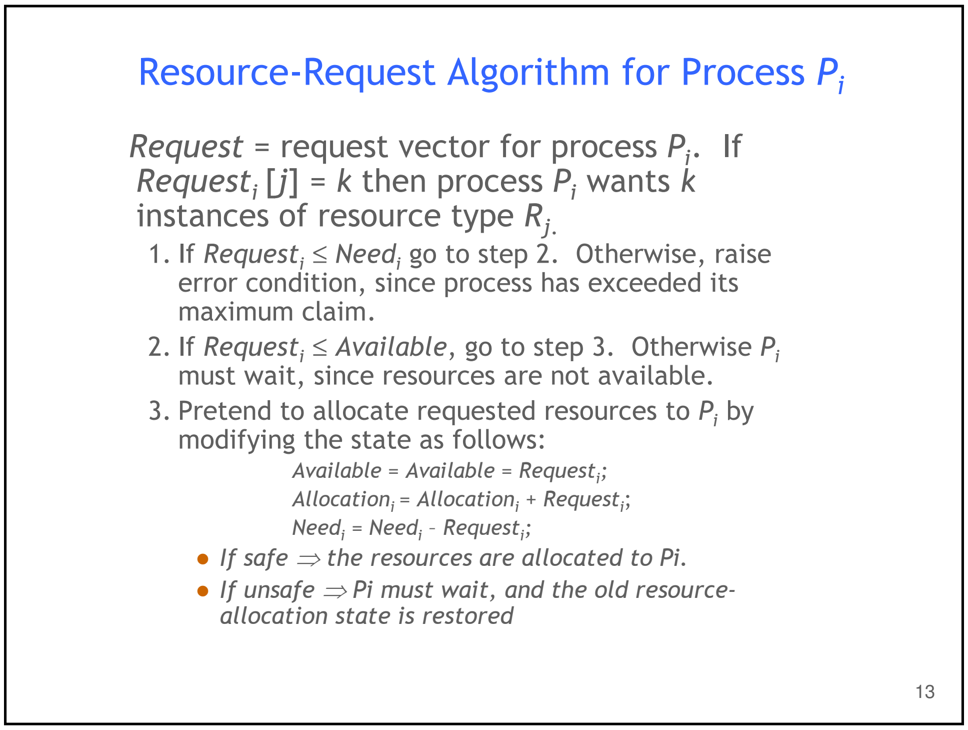
Let n = number of processes, and m = number of resources types.

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

Need [i,j] = *Max*[i,j] – *Allocation* [i,j].

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length m and n, respectively. Initialize:
   Work = *Available*
   Finish [i] = *false* for i - 1,3, …, n.
2. Find and i such that both:
   (a) *Finish* [i] = *false*
   (b) $Need_i \leq Work$
   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.



Resource-Request Algorithm for Process P_i

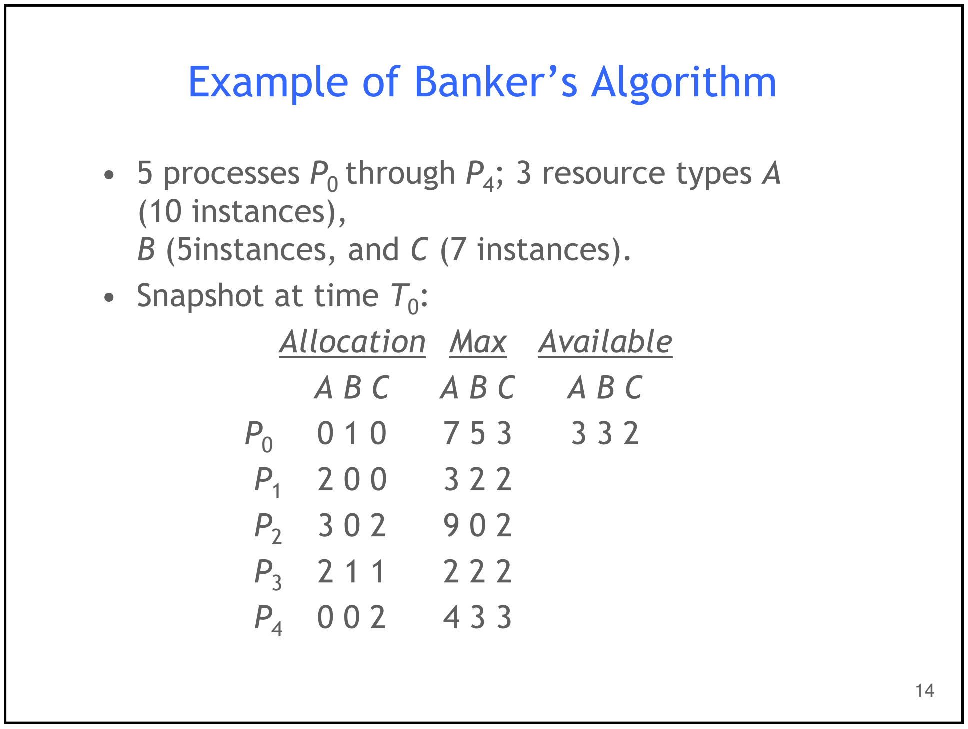
Request = request vector for process P_i. If *Request*$_i$ [j] = k then process P_i wants k instances of resource type R_j.

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. 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*$_i$;

   - *If safe* $\Rightarrow$ *the resources are allocated to Pi.*
   - *If unsafe* $\Rightarrow$ *Pi must wait, and the old resource-allocation state is restored*

13



Example of Banker's Algorithm

- 5 processes P_0 through P_4; 3 resource types *A* (10 instances),
  B (5instances, 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	

14

Example (Cont.)

- The content of the matrix. Need is defined to be 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

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

15

Example P_1 Request (1,0,2) (Cont.)

- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true.

	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 1	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 <P1, P3, P4, P0, P2> satisfies safety requirement.
- Can request for (3,3,0) by P4 be granted?
- Can request for (0,2,0) by P0 be granted?

16

Deadlock Detection

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

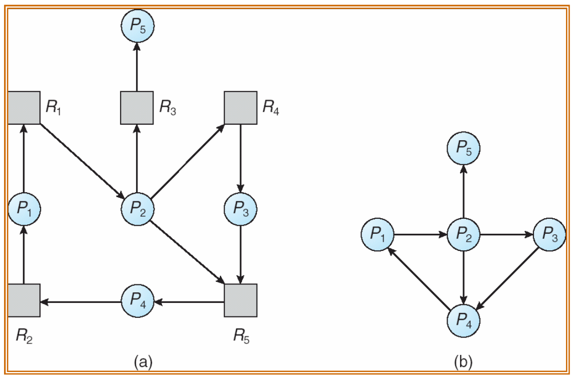
17

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.
- 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.

18

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph   Corresponding wait-for graph

Several Instances of a Resource Type

- *Available:* A vector of length *m* indicates the number of available 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 $Allocation_i \neq 0$, 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.

Detection Algorithm (Cont.)

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

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*.

23

Example (Cont.)

- P_2 requests an additional instance of type *C*.

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

- 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.

24

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.

25

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?

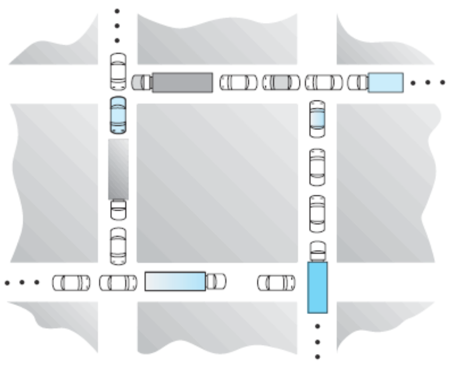
26

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.

27

Exercise



a. Show that the four necessary conditions for deadlock indeed hold in this example.

b. State a simple rule for avoiding deadlocks in this system.

28

Solution

a. The four necessary conditions for a deadlock are (1) mutual exclusion; (2) hold-and-wait; (3) no preemption; and (4) circular wait. The mutual exclusion condition holds as only one car can occupy a space in the roadway. Hold-and-wait occurs where a car holds onto their place in the roadway while they wait to advance in the roadway. A car cannot be removed (i.e. preempted) from its position in the roadway. Lastly, there is indeed a circular wait as each car is waiting for a subsequent car to advance. The circular wait condition is also easily observed from the graphic.

b. A simple rule that would avoid this traffic deadlock is that a car may not advance into an intersection if it is clear they will not be able to immediately clear the intersection.

29

Exercise

Consider the dining-philosophers problem where the chopsticks are placed at the center of the table and any two of them could be used by a philosopher. Assume that requests for chopsticks are made one at a time. Describe a simple rule for determining whether a particular request could be satisfied without causing deadlock given the current allocation of chopsticks to philosophers.

30

Solution

Answer: The following rule prevents deadlock: when a philosopher makes a request for the first chopstick, do not satisfy the request only if there is no other philosopher with two chopsticks and if there is only one chopstick remaining.

31

Any Questions?

32

Reading Assignment

- Read chapter 7 from Silberschatz.

Acknowledgements

- "Operating Systems Concepts" book and supplementary material by Silberschatz, Galvin and Gagne.