CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 25

Deadlock

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

- Does a cycle in a resource allocation graph signify “circular wait”? Only if there is only one instance of a resource
- Safe state idea: What about resources held by processes that have already run? they have been released and are thus available.
- Does the MAC spinning wheel indicate a deadlock?
- Are application hangs caused by deadlocks? OS timer perhaps 5 sec

Chapter 7: Deadlocks

- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
  - Deadlock Prevention
  - 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

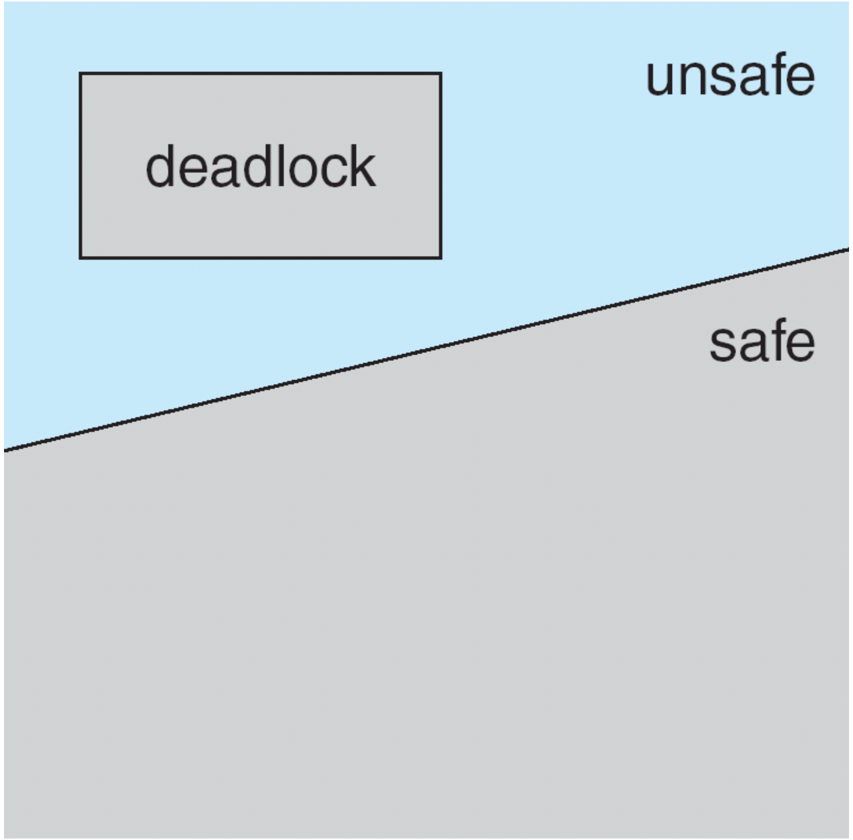
Safe State

- 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 sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the systems such that 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 $j < I$
- That is:
  - 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

Basic Facts

- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

Safe, Unsafe, Deadlock State



Example A: 12 Units available in the system

	Max need	Current holding
P0	10	5
P1	4	2
P2	9	2

At T0:
9 units allocated
3 units available

A unit could be a drive, a block of memory etc.

- At time **T0 is** the system is in a safe state?
  - Try sequence <P1, P0 , P2>
  - P1 can be given 2 units
  - When P1 releases its resources; there are 5 units
  - P0 uses 5 and subsequently releases them (# 10 now)
  - P2 can then proceed.
- Thus <P1, P0 , P2> is a safe sequence, and at T0 system was in a safe state

Example B: 12 Units available in the system

	Max need	Current holding
P0	10	5
P1	4	2
P2	9	2+1

Before T1:
3 units available

At T1:
2 units available

- At time **T1,** P2 is allocated 1 more units. Is that a good decision?
  - Now only P1 can proceed.
  - When P1 releases its resources; there are 4 units
  - P0 needs 5 and P2 needs 6. Deadlock.
    - **Mistake** in granting P2 additional units
- The state at **T1** is not a safe state.

Avoidance Algorithms

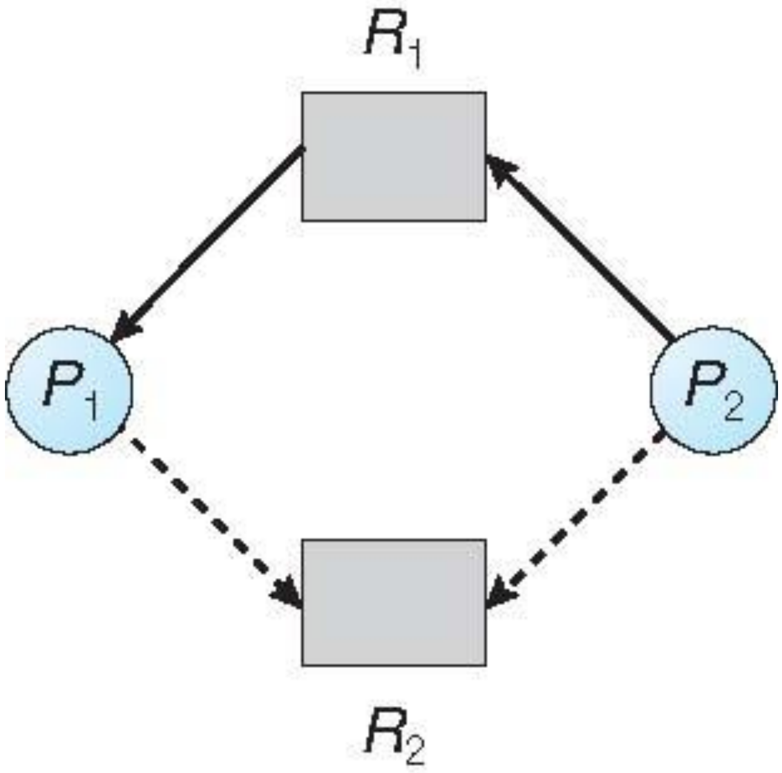
- Single instance of a resource type
  - Use a resource-allocation graph scheme

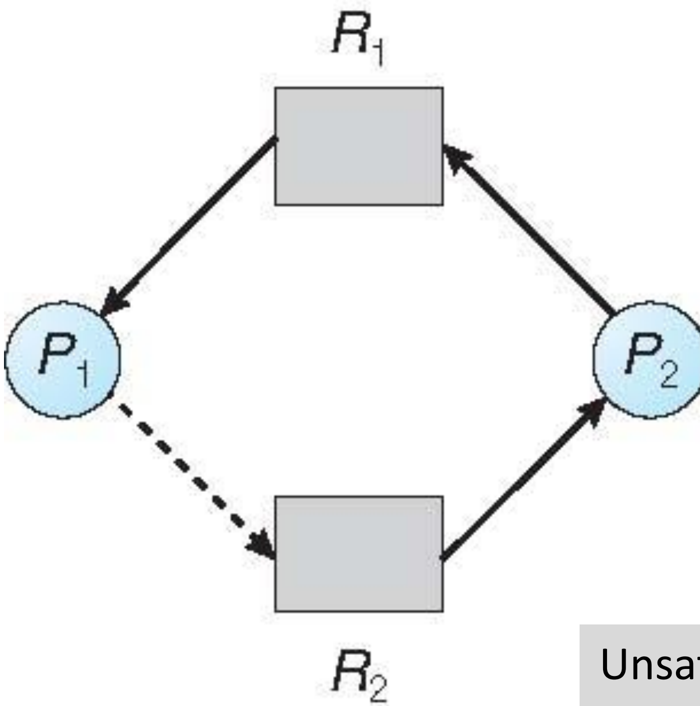
- Multiple instances of a resource type
  - Use the banker's algorithm

Resource-Allocation Graph Scheme

- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system

Resource-Allocation Graph





Unsafe state

Suppose *P2* requests *R2.* Although *R2* is currently free, we cannot allocate it to *P2,* since this action will create a cycle getting system in an unsafe state. If *P1* requests *R2,* and *P2* requests *R1,* then a deadlock will occur.

Colorado State University

Resource-Allocation Graph Algorithm

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

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

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

Processes vs resources:

- **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

$$Need[i,j] = Max[i,j] – Allocation[i,j]$$

Colorado State University

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

Colorado State University

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$_i$* ≤ *Need$_i$*** go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If ***Request$_i$* ≤ *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:

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

   - 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 of 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

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

Colorado State University

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

Colorado State University

Ex: Assume now P_1 Requests (1,0,2)

- Check that Request ≤ Available
  - (1,0,2) ≤ (3,3,2) ⇒ true. Check for safety after pretend allocation. P1 allocation would be (2 0 0) + (1 0 2)

	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.

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.

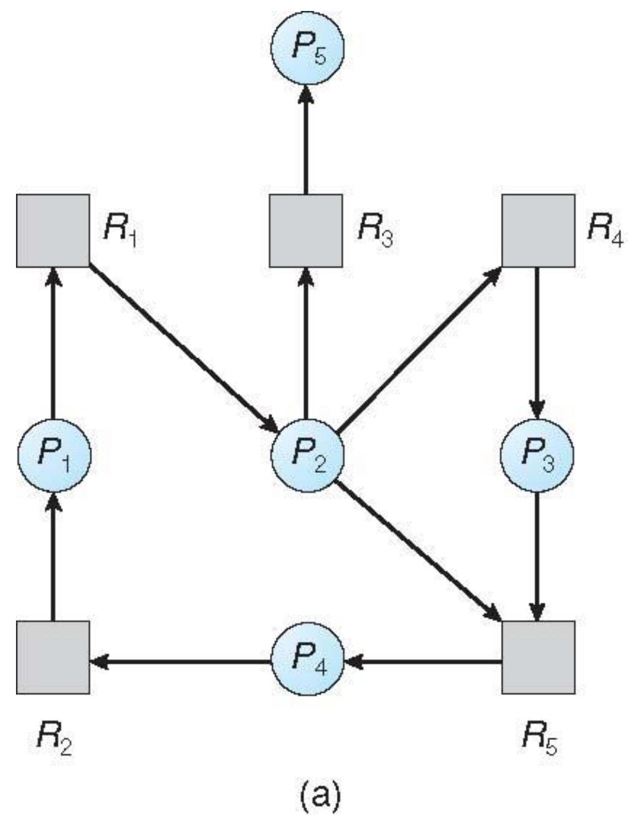
Deadlock Detection

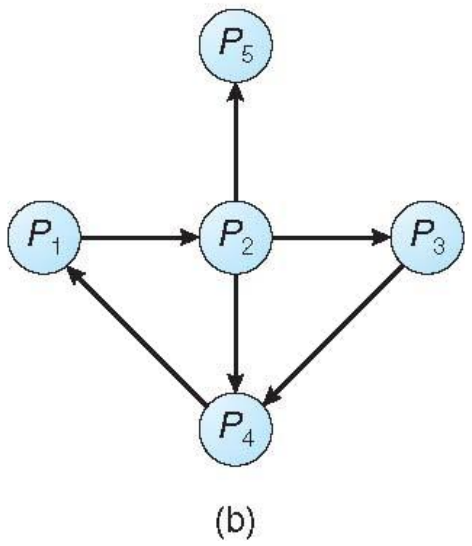
- Allow system to enter deadlock state
- Detection algorithm
  - Single instance of each resource:
    - wait-for graph
  - Multiple instances:
    - detection algorithm (based on Banker's algorithm)
- Recovery scheme

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 and Wait-for Graph





Resource-Allocation Graph   Corresponding wait-for graph

3 cycles. Deadlock.

Colorado State University

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.