Operating System

Nguyen Tri Thanh
ntthanh@vnu.edu.vn

Review

Which is correct about race condition?

A. Happen even when there is only once process
B. Happen when multiple processes use a shared resource concurrently
C. Happen when multiple processes use a resource sequentially
D. Happen when there are multiple processes in the system

Review

Which is incorrect about the Peterson’s solution?

A. It satisfies all the conditions of critical section
B. It is easy to control even the number of processes is above 2
C. It is difficult to control
D. It is complicated when the number of processes is above 2

Review

Which of the following is the most correct about critical section?

A. A code snippet that operates on a global variable
B. A code snippet that operates on a resource
C. A code snippet that operates on a global resource
D. A code snippet that operates on a shared resource

Review

How many conditions for resolving critical section are there ?

A. 1
B. 2
C. 3
D. 4

Review

Which is incorrect about the conditions of critical section?

A. The progress condition utilizes the resource effectively
B. The exclusive condition removes race condition
C. The exclusive condition ensures processes to use a shared resource sequentially
D. The bounded waiting condition allows a process to use a shared resource several consecutive times

Question

Which is the purpose of the second condition of critical section?

A. It reduces the waiting time of requested processes
B. It ensures the correct use of the shared resource
C. It makes the algorithm more complicated to implement
D. It makes the algorithm less complicated to implement

Question

Which is the purpose of the third condition of critical section?

A. It supports the priority of processes
B. It ensures the correct use of the shared resource
C. It utilizes the shared resource effectively
D. It makes sure no process is in its critical section forever

Review

Which is incorrect about the semaphore?

A. Semaphore is an implementation of critical section
B. Semaphore does not guarantee the conditions of critical section
C. A semaphore usually includes an integer variable
D. Semaphore has atomic operators

Review

How many types the semaphore are there?

A. 1
B. 2
C. 3
D. 4

Review

Which of the following is correct about counting semaphore?

A. The value of the semaphore is 0 or 1

B. The same as binary semaphore

C. The value of the semaphore variable can be above 1

D. The value of the semaphore variable can nerver be below 0

Review

Which of the following is the most suitable use for counting semaphore?

A. Use for shared resources with a single instance
B. Use for shared resources with 2 instances
C. Use for shared resources with any instances
D. Use for shared resources with multiple instances

Deadlock

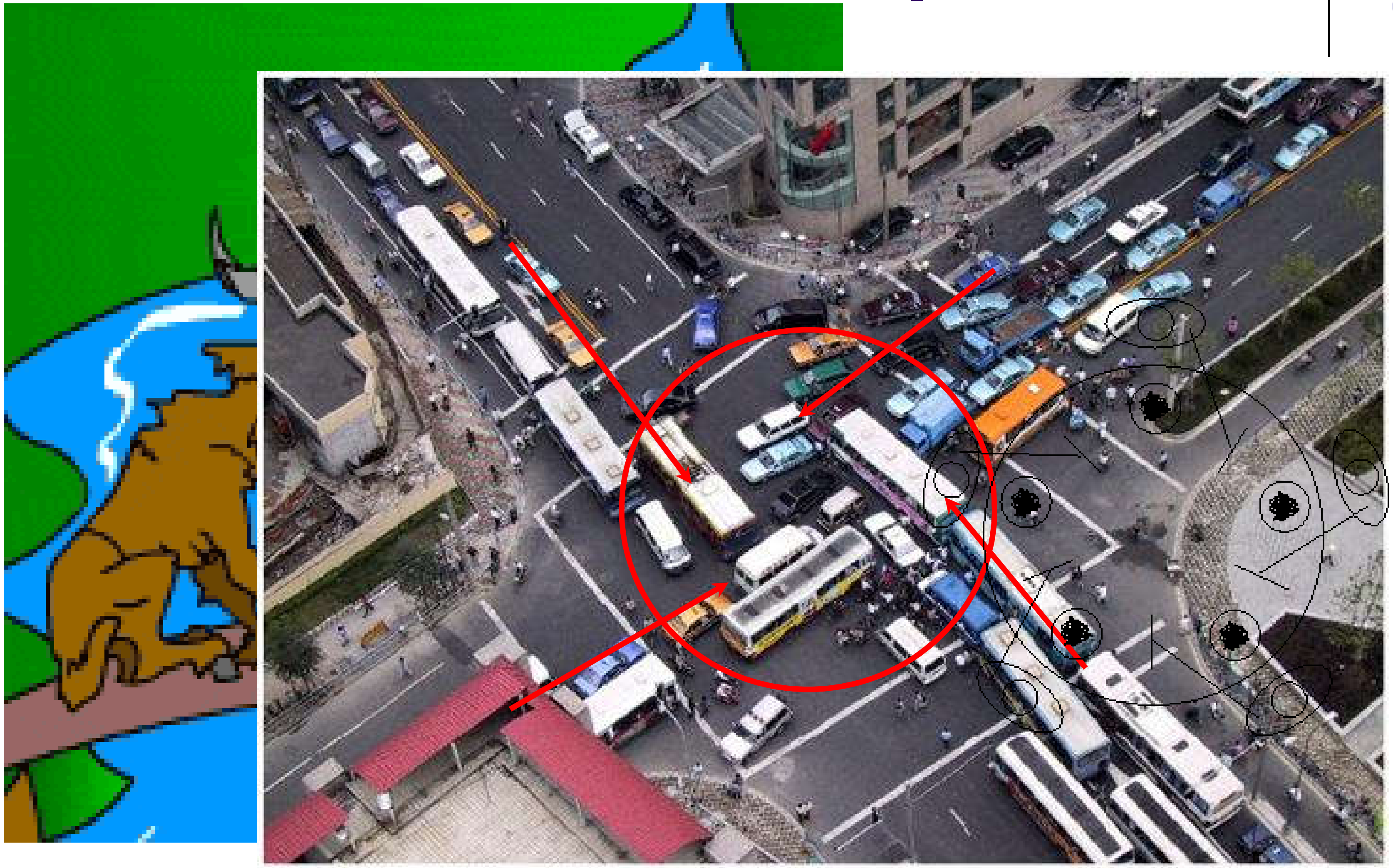
Objectives

- Introduce what a deadlock is
- Introduce methods of handling deadlocks
- Implement deadlock handling algorithms

Reference

- Chapter 7 of Operating System Concepts

Deadlock examples



Definition of deadlock

- A set of blocked processes each
  - holding a resource and
  - waiting to acquire a resource held by another process in the set
- There must be a circular wait in this set

Deadlock example (cont'd)

- Process A:

{

…

Lock file F_1;

...

Open file F_2;

…

Unlock F_1;

}

- Process B

{

…

Lock file F_2;

...

Open file F_1;

…

Unlock F_1;

}

Question

When does the deadlock happen?

A. A gets F1 and waits for F2

B. A gets F2 and waits for F1 and B waits for F1

C. A gets F1 and waits for F2 and B gets F2 and waits for F1

D. A gets F1 and F2 and B waits for F2

Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously
  - **C1: Mutual exclusion**
  - **C2: Hold and wait** holding one resource, waiting other resources held by another
  - **C3: No preemption** only process has right to release its holding resources
  - **C4: Circular wait** there exists a set $\{P_0, P_1, \ldots, P_0\}$ of processes:
    - P_0 is waiting for a resource that is held by P_1,
    - P_1 is waiting for a resource that is held by P2, …
    - P_n is waiting for a resource that is held by P_0.

System Model

- Resource types $R_1, R_2, \ldots, R_m$
  - *shared variables, memory space, I/O devices,*
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - request
  - use
  - release

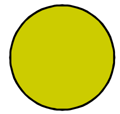
Resource-Allocation Graph

A set of vertices *V* and a set of edges *E*.

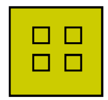
- V is partitioned into two types
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_i \rightarrow R_j$
- assignment edge – directed edge $R_j \rightarrow P_i$

Resource-Allocation Graph (Cont'd)

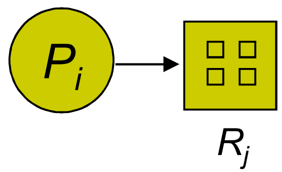
- Process



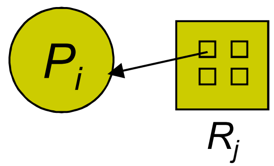
- Resource Type with 4 instances



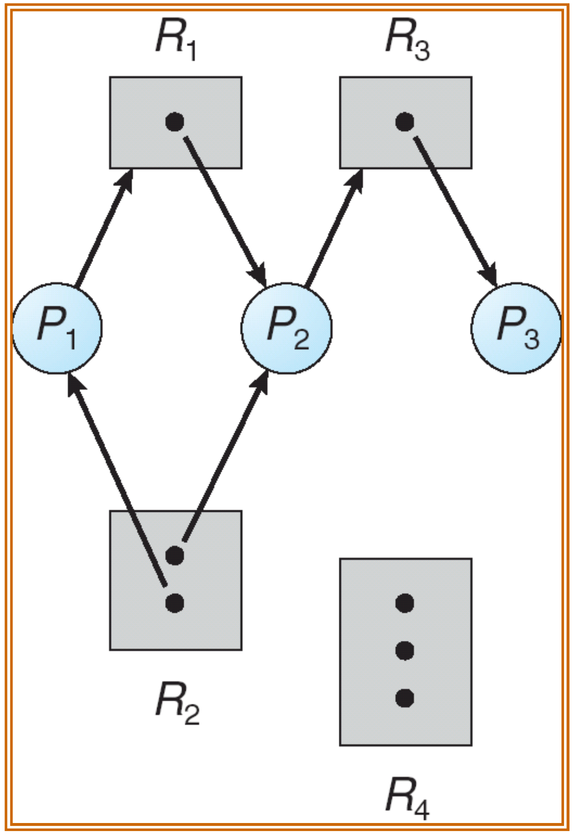
- P_i requests instance of R_j



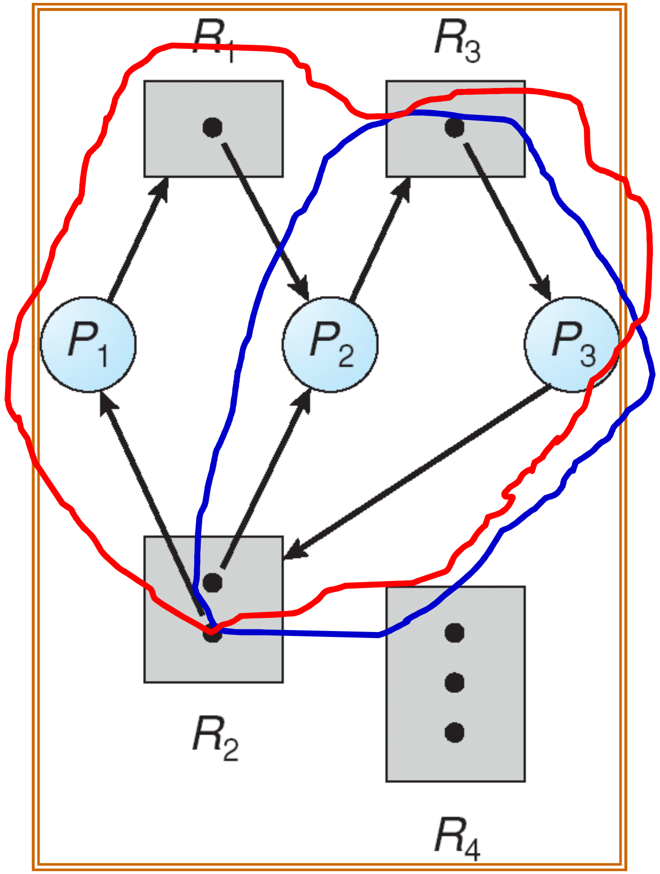
- P_i is holding an instance of R_j



Example of a RAG

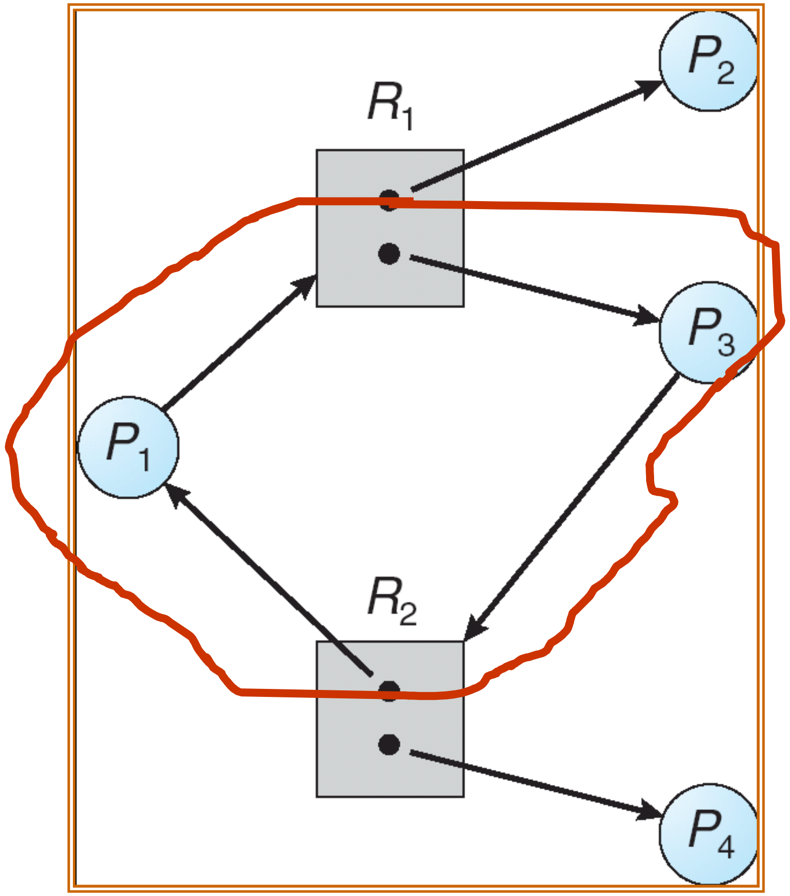


RAG With A Deadlock



- When P_3 asks for R_2
- There are two circulars
  - $P_1 \rightarrow R_1 \rightarrow P_2 \rightarrow R_2 \rightarrow P_3 \rightarrow R_3 \rightarrow P_1$
  - $P_2 \rightarrow R_3 \rightarrow P_3 \rightarrow R_2 \rightarrow P_2$
- Set of P_1, P_2, P_3 is deadlock

Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock.
  If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock.
  - if several instances per resource type, possibility of deadlock.

Deadlock handling

Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state
  - Deadlock prevention, deadlock avoidance
- Allow the system to enter a deadlock state and then recover
  - Deadlock detection and recovery
- Ignore the problem and pretend that deadlocks never occur in the system
  - used by most operating systems, including UNIX.

Deadlock Prevention

- The method prevents at least one of the four deadlock conditions from occurring
- This method is classified as a static method

Deadlock Prevention

- **C1: Mutual Exclusion**
  - In some situations, this condition is required
  - Not feasible to make this NOT to happen

Deadlock Prevention

- **C2: Hold and Wait**
  - Solution
    - must guarantee that whenever a process requests a resource, it does not hold any other resources, or
    - require process to request and be allocated all its resources before it begins execution
  - low resource utilization; starvation possible.

Deadlock Prevention (Cont'd)

- **C3: No Preemption**
  - If a process holding some resources requests another resource that cannot be immediately allocated to it,
    - then all resources currently being held are released
    - released resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources and the new requesting ones

Deadlock Prevention (Cont'd)

- **C4: Circular Wait**
  - impose a total ordering of all resource types and
  - require that each process requests resources in an increasing order of enumeration

Question

How many conditions for a dead lock to happen are there?

A. 2
B. 3
C. 4
D. 5

Question

When does a deadlock happen?

A. any of the 4 conditions occur
B. any two of the 4 conditions occur
C. any 3 of the 4 conditions occur
D. all the 4 conditions occur

Deadlock avoidance

Deadlock Avoidance

- This method requires additional information to decide resource allocation so that deadlock will not happen
  - each process has to register the number of each required resource types as additional information
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition

Deadlock Avoidance

- Deadlock avoidance algorithms check the state of resource-allocation to decide allocation
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes

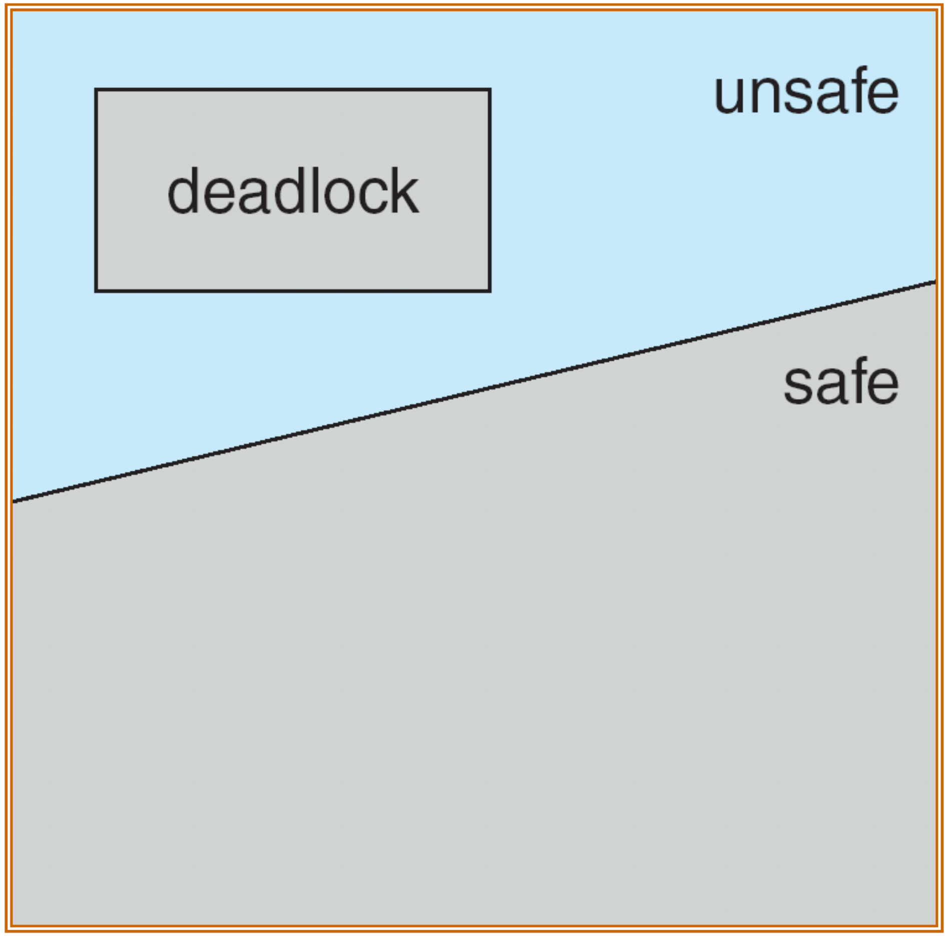
Safe State

- System is in safe state if a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes exists
  - P_i can be satisfied by currently available resources + resources held by all the P_j, with $j < i$
  - processes terminate in the above order

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 system has 12 tapes, and 3 processes P_0, P_1, P_2 with corresponding requests:
  - P_0 requests at most 10 tapes
  - P_1 requests at most 4 tapes
  - P_2 requests at most 9 tapes
- At t_0, P_0 has 5 tapes, P_1 and P_2 each has 2 tapes
  - 3 tapes available

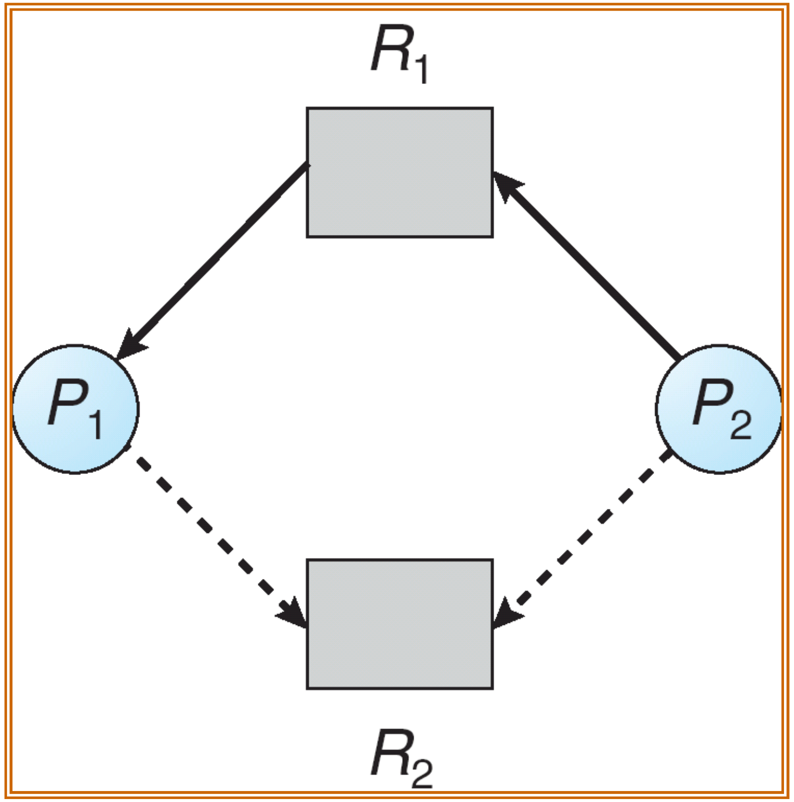
Avoidance algorithms

- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker's algorithm

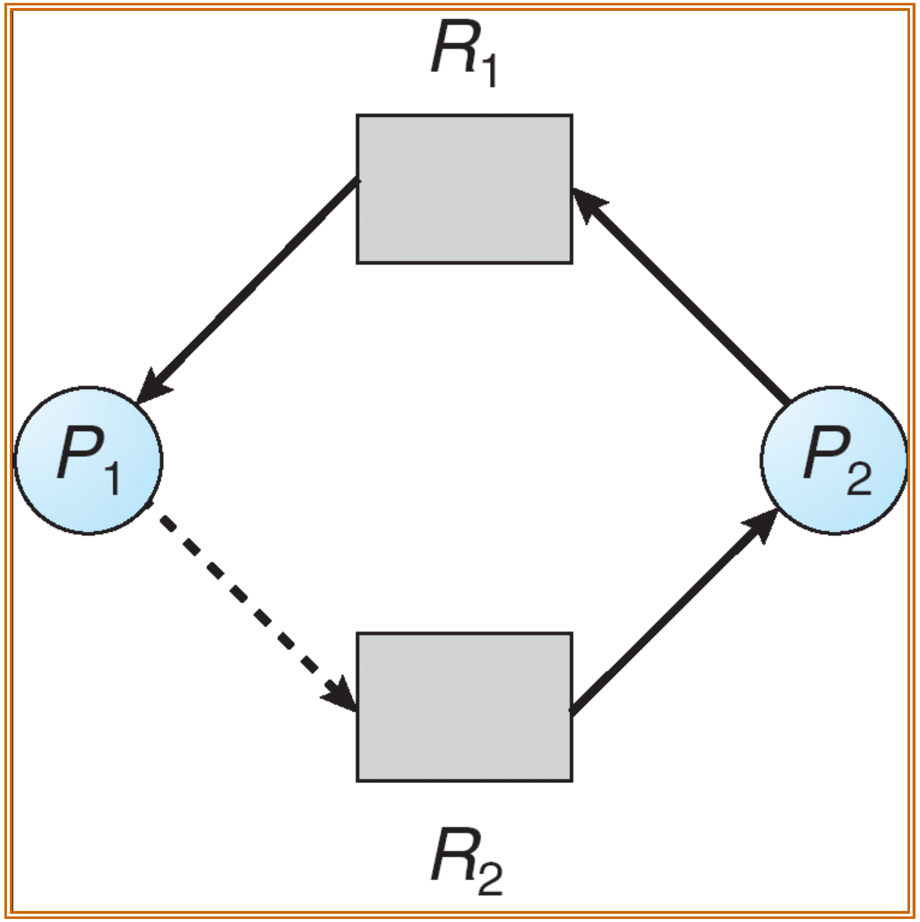
RAG Algorithm convention

- *Claim edge $P_i \rightarrow R_j$*
  - process P_j may request resource R_j
  - presented as a dash line
  - Claim edge converts to request edge when a process requests a resource
- Request edge becomes an assignment edge when the resource is assigned to it
- 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 In Resource-Allocation Graph



Resource-Allocation Graph Algorithm

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

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

Data Structures for the Banker's Algorithm

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

- ***Available***: Vector of length m
  - Available [j] = k: there are k instances of resource type R_j available
- ***Max***: n x m matrix
  - *Max* [i,j] = k: P_i may request at most k instances of R_j
- ***Allocation***: n x m matrix
  - Allocation[i,j] = k: P_i is allocated k instances of R_j

Data Structures for the Banker's Algorithm

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

- ***Need***: n x m matrix
  - $Need[i,j] = k$: P_i may need k more instances of R_j
  - $Need\ [i,j] = Max[i,j] – Allocation\ [i,j]$
- Let ***Work*** and ***Finish*** be vectors of length m and n, respectively
- Let $A=(A_1, A_2, \ldots, A_n)$, $B=(B_1, B_2, \ldots, B_n)$
- Define $A \leq B$ if only if $A_i \leq B_i$, $\forall 1 \leq i \leq n$

Safety/Banker Algorithm

1. Initialize

   Work = *Available*

   Finish [*i*] = *false* for $i = 0, 1, \ldots, n-1$

2. Find an *i* that satisfies 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

Question

Which of the following is correct about the *Work* variable in the algorithm?

A. It stores the available resources when each process finishes
B. It is a redundant variable
C. It stores the state of the system
D. It stores possible resources for each process

Question

Which of the following is the most correct about banker's algorithm?

A. it detects the unsafe state of the system
B. it detects the deadlock state of the system
C. it detects the safe sequence of the system
D. it detects the available resources

Example of Banker's Algorithm

- 5 processes: P_0 - $P_{4;}$ 3 resource types
  - A (10 instances), B (5 instances), and C (7 instances)
- 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	

Example (Cont'd)

- Matrix *Need* = *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 or not?
  - sequence < P_1, P_3, P_0, P_2, P_4> satisfies safety criteria.

Example

A system has 12 tapes, and 3 processes P_0, P_1, P_2 with corresponding requests:

	Max request	Current Allocation
P_0	10	5
P_1	4	2
P_2	9	2

- At t_0, the system is in safe state
  - The sequence <P_1, P_0, P_2> is a safe sequence
- At t_1, P_2 requests1 more tape (is it safe?)
  - the system is in unsafe state
  - it is wrong to allocate a tape for P_2

Resource-Request Algorithm

- Resource-request algorithm
  - another algorithm to avoid unsafe state
- Additional data structure
  - *Request* = request vector for process P_i
  - $Request_i[j] = k$: process P_i wants k instances of R_j

Resource-Request Algorithm

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 P_i.*
- *If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored*

Question

Which of the following is correct about resource-request algorithm?

A. it detects the unsafe state of the system
B. it detects the deadlock state of the system
C. it detects the safe sequence of the system
D. it detects the safe sequence of the system if the request is granted

Example: P_1 Request (1,0,2)

- Request $\leq$ Available $((1,0,2) \leq (3,3,2) \Rightarrow$ 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	

- $< P_1, P_3, P_0, P_2, P_4 >$ is a safe sequence
  - Can request for (1,0,0) by P_4 be granted?
  - Can request for (0,2,0) by P_0 be granted?

Deadlock detection

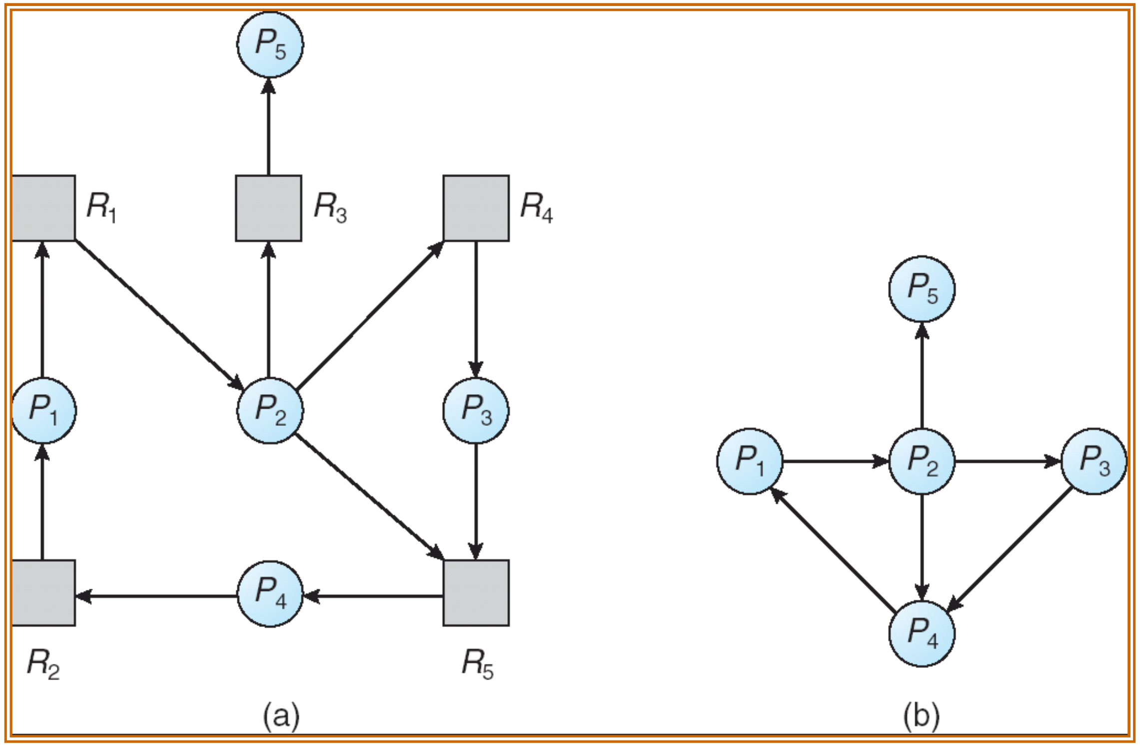
Deadlock Detection

- Allow system to enter deadlock state
- Use detection algorithms
- Recover from deadlock

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
  - 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

Several Instances of a Resource Type

- ***Available***: A vector of length *m*
  - number of available resources of each type
- ***Allocation***: An *n* x *m* matrix
  - number of resources of each type currently allocated to each process
- ***Request***: An *n* x *m* matrix
  - 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

Let *Work* and *Finish* be vectors of length *m* and *n*, respectively

1. Initialize:

   (a) *Work* = *Available*

   (b) For $i = 1,2, \ldots, n$,

   if $Allocation_i \neq 0$ OR $Request_i \neq 0$, 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'd)

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 $O(m \times n^2)$ operations to detect whether the system is in deadlocked state.

Example of Detection Algorithm

- Processes P_0 - P_4; resources (numbers)
  - A (7), *B* (2), and *C* (6)
- Snapshot at time T_0 (deadlock?)

	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_1, P_3, P_4> will result in *Finish*[*i*] = true for all *i*.

Example (Cont'd)

- 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 (deadlock? processes in deadlock?)
  - Can reclaim resources held by process P_0, but insufficient resources to fulfill other processes
  - Deadlock exists, consisting of processes P_1, P_2, P_3, and P_4

Question

- Which of the following is correct about deadlock detection algorithm?
  A. it only detects the unsafe state of the system
  B. all the processes in the system are in the deadlock when it detects a deadlock
  C. it can only detect the deadlock not the processes involved in the deadlock
  D. it can detect deadlock as well as the involved processes

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
  - would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes
- Abort each process until the deadlock is removed
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and/or 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?

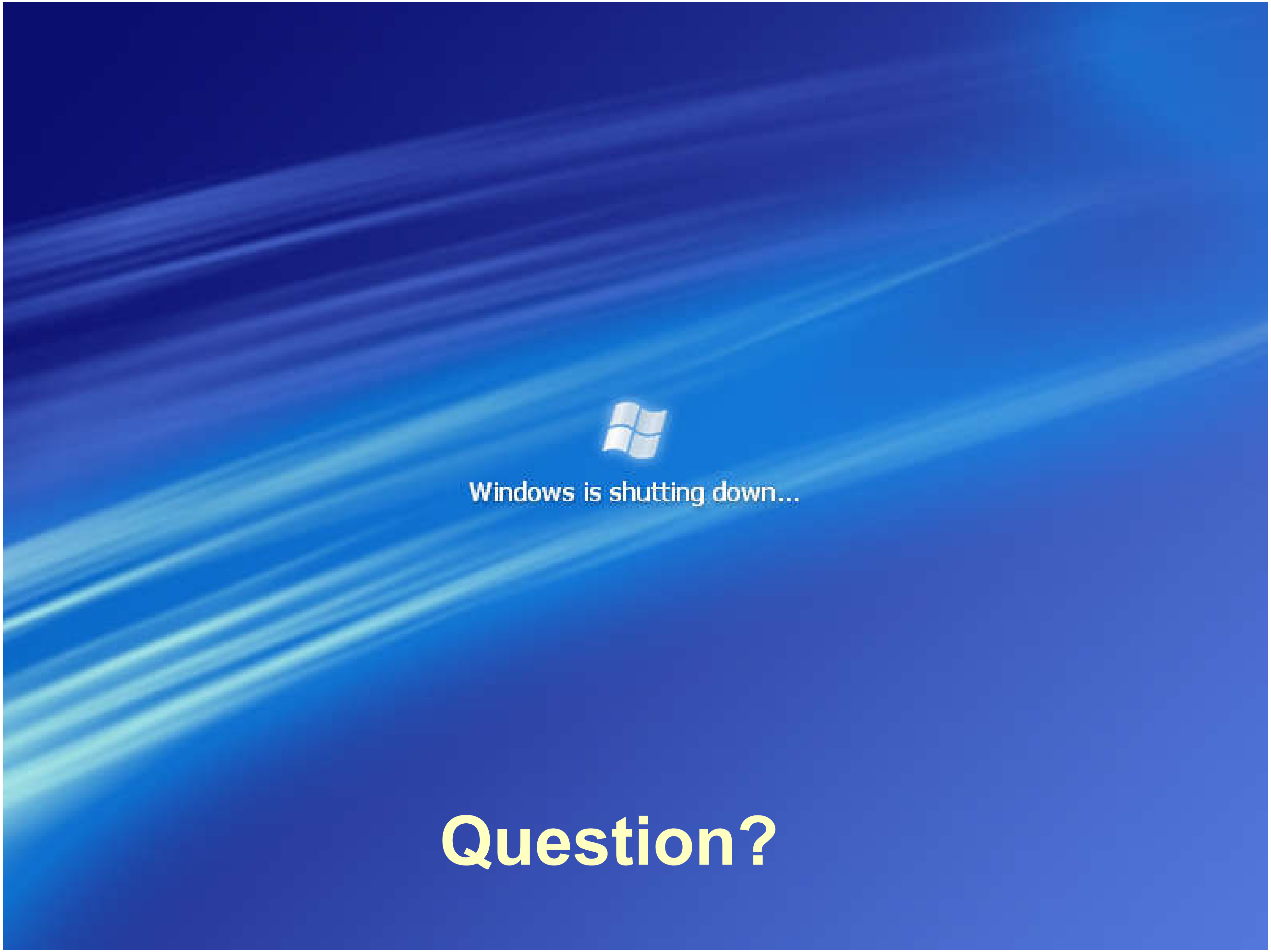
Recovery from Deadlock: Resource Preemption

- 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

Discussion

- For each abort condition, discuss which process will be selected to be cancelled
  - 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?

End of chapter



Question?