北京邮电大学软件学院
School Of software Engineering Of BUPT

Operating Systems

Lecture 5 Deadlocks

Jinpengchen
Email: jpchen@bupt.edu.cn

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

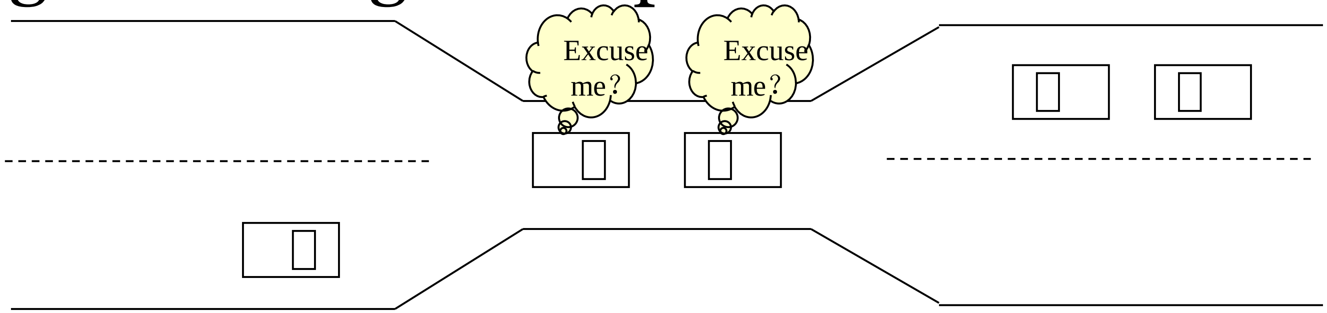
The Deadlock Problem

- The Deadlock Problem
  - Deadlock Situation : A set of blocked processes is in a deadlock state when each holds a resource and waits to acquire a resource held by another process in the set.
  - Example 1
    - ✓ System has 2 disk drives
    - ✓ P1 and P2, each holds one disk drive and each needs another one
  - Example 2
    - ✓ semaphores A and B, initialized to 1

P0	P1
wait (A);	wait(B)
wait (B);	wait(A)

The Deadlock Problem

- Bridge Crossing Example



  - Traffic only in one direction
  - Each section of a bridge can be viewed as a resource
  - If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback)
  - Several cars may have to be backed up if a deadlock occurs
  - Starvation is possible
  - Note – Most OSes do not prevent or deal with deadlocks

Deadlock Characterization

- Why did the deadlock happen?
  - ✓ Insufficient resources
  - ✓ Unreasonable execution order

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock Characterization

- Necessary Conditions
  - Deadlock can arise if four conditions hold simultaneously.
    - ✓Mutual exclusion(互斥):
      - only one process at a time can use a resource.
    - ✓Hold and wait(持有并等待):
      - a process holding at least one resource is waiting to acquire additional resources held by other processes.
    - ✓No preemption(不剥夺):
      - a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Deadlock Characterization

- Necessary Conditions
  - ✓Circular wait(循环等待):

    - there exists a set $\{P_0, P_1, \cdots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, $\cdots$, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

Deadlock Characterization

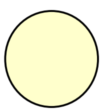
- System Model
  - A system consists of a finite number of resources
  - The resources are partitioned into several types, each consisting of some number of identical instance.
    - ✓ physical resources: CPU cycles, memory space, I/O devices
    - ✓ logical resources: files, semaphores, and monitors
  - System model
    - ✓ Resource types R1 , R2 , . . . , Rm
    - ✓ Each resource type Ri has Wi instances.
    - ✓ Each process utilizes a resource as follows:
      - request: may wait until it can acquire the resource
      - use
      - release

Deadlock Characterization

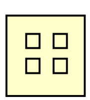
Resource-Allocation Graph

- System resource-allocation graph: A directed graph
  - ✓ A set of vertices V and a set of edges E.
  - ✓ V is partitioned into two types:

  \- P = {P1, P2,…, Pn}, the set consisting of all the processes in the system.

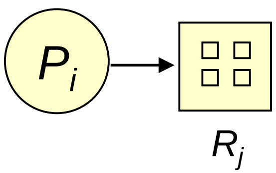
 : Process

\- R = {R1, R2,…, Rm}, the set consisting of all resource types in the system.

 : Resource Type with 4 instances

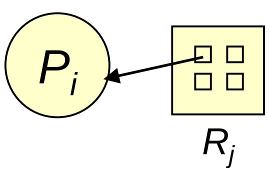
Deadlock Characterization

- Resource-Allocation Graph
  - ✓ E is partitioned into two types.
    - request edge(请求边) – directed edge Pi→Rj



: Pi requests an instance of Rj

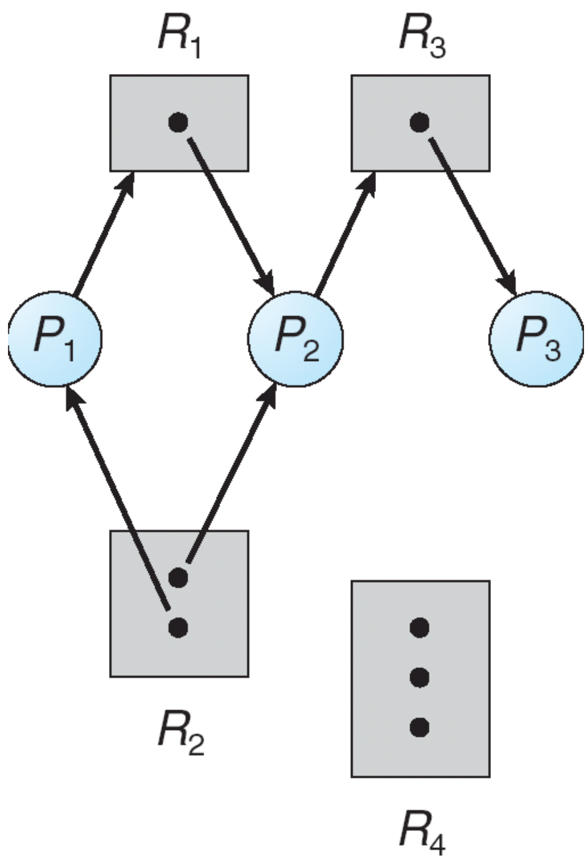
- assignment edge(分配边) – directed edge Rj→Pi



: Pi is holding an instance of Rj

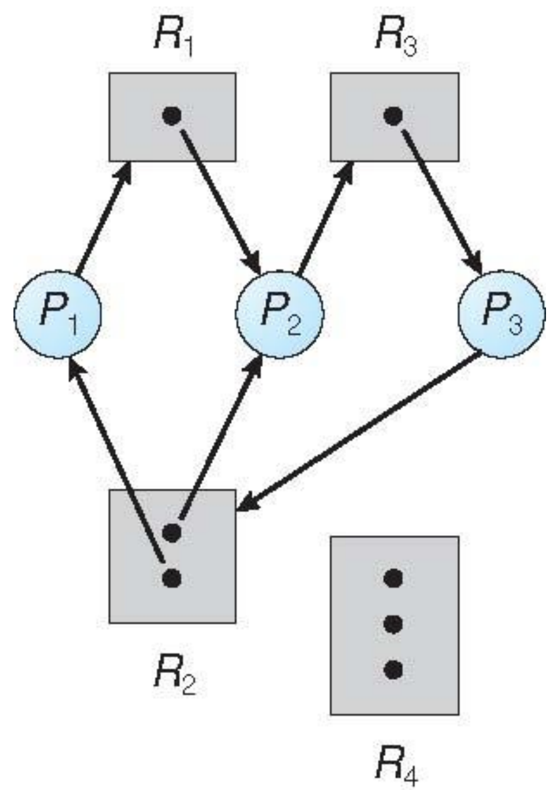
Deadlock Characterization

- An Example of a Resource Allocation Graph



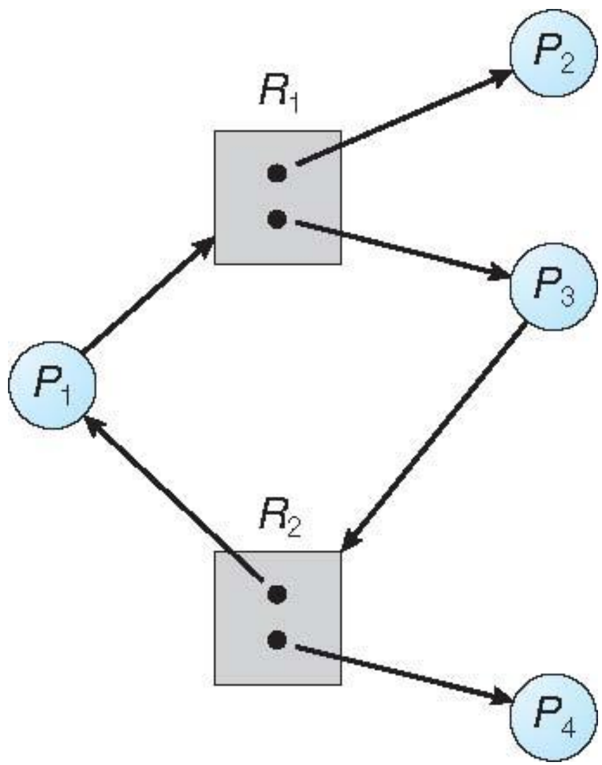
Deadlock Characterization

- Example of a Resource Allocation Graph with a Deadlock



Deadlock Characterization

- A Graph with a Cycle but No Deadlock



Deadlock Characterization

- 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.

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Methods for Handling Deadlocks

- 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 from deadlock
  - Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

request	→	allocation	→	runtime
↑		↑		↑
prevention		avoidance		detection and recovery

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock Prevention

- Deadlock prevention provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.
- Mutual Exclusion
  - Not required for sharable resources (read-only files); must hold for nonsharable resources. (printer)
  - In general, we cannot deny the mutual-exclusion condition

Deadlock Prevention

- Hold and Wait
  - must guarantee that whenever a process requests a resource, it does not hold any other resources.
    - ✓ Require process to request and be allocated all its resources before it begins execution, or
    - ✓ Allow process to request resources only when the process has none.
  - Disadvantage:
    - ✓ Low resource utilization;
    - ✓ Starvation possible.

Deadlock Prevention

- No Preemption
  - If a process that is holding some resources, requests another resource that cannot be immediately allocated to it, then all resources currently being held are preempted.
    - ✓ Preempted 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, as well as the new ones that it is requesting.
  - preempt the desired resources from the waiting process and allocate them to the requesting process
    - ✓ if the resource are neither available nor held by a waiting process, the requesting process must wait. While waiting, some of its resources may be preempted by other requesting process
    - ✓ a process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted.

Deadlock Prevention

- Circular Wait
  - impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.
    - ✓ always in an increasing order
    - ✓ may release some higher ordered resource before requesting lower ordered resource

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock Avoidance (死锁避免)

- Execute a deadlock avoidance algorithm to ensure there can never be a circular-wait condition.

Banker’s Algorithm

Deadlock Avoidance (死锁避免)

- Requires that the system has 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.

Deadlock Avoidance (死锁避免)

- An Example
  - Total resources 12; 3 processes
  - Snapshot at time t_0

	Max	Allocation	Need	Available
P1	10	5	5	3
P2	4	2	2	
P3	9	2	7	

Deadlock Avoidance (死锁避免)

Safe State (安全状态)

- 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 <P1, P2, …, Pn> of all the processes
- Sequence <P1, P2, …, Pn> is safe if for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < i

  That is:

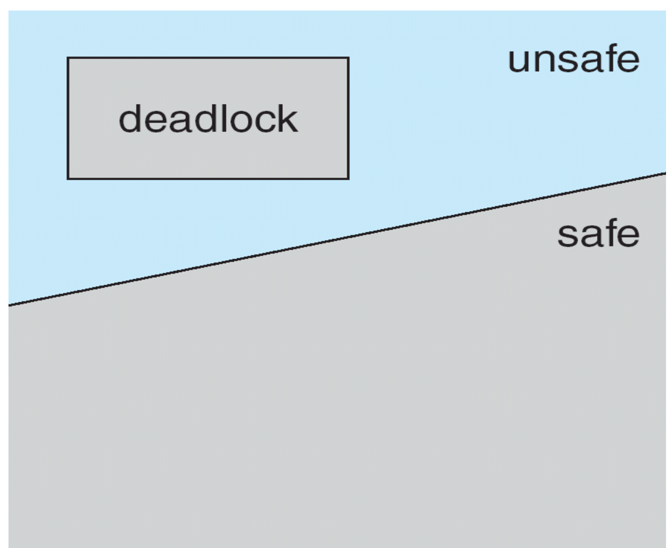
  ✓ If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.

  ✓ When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

  ✓ When Pi terminates, Pi+1 can obtain its needed resources, and so on.

Deadlock Avoidance (死锁避免)

- 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.



Deadlock Avoidance (死锁避免)

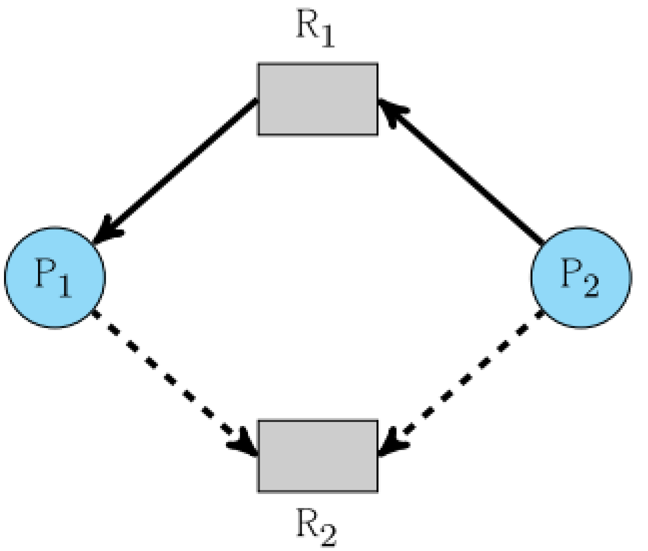
- Avoidance algorithms
  - Single instance of a resource type
    - ✓ Use a resource-allocation graph
  - Multiple instances of a resource type
    - ✓ Use the banker’s algorithm

Deadlock Avoidance (死锁避免)

- Resource-Allocation Graph Scheme
  - Resource-Allocation Graph
    - ✓ Claim edge(需求边) Pi → Rj
      - indicated that process Pj may request resource Rj;
      - represented by a dashed line
    - ✓ Claim edge converts to request edge when a process requests a resource
    - ✓ Request edge converts 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

Deadlock Avoidance (死锁避免)

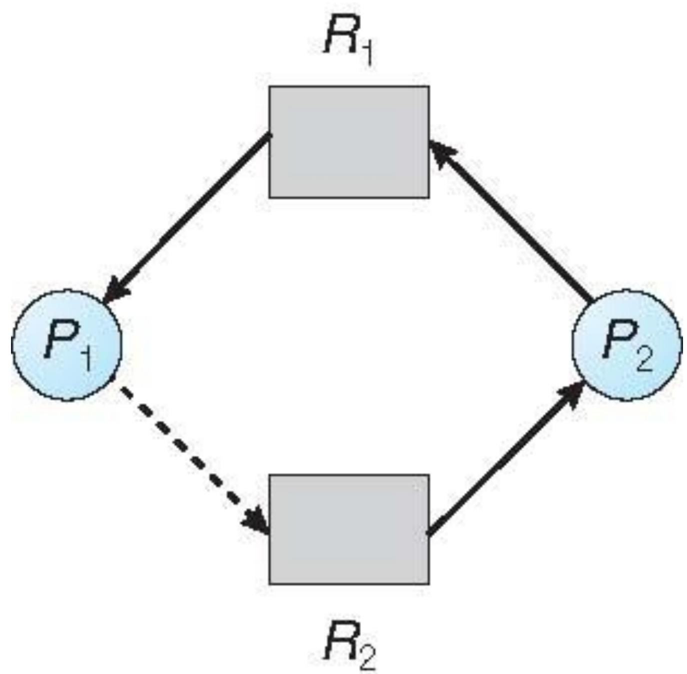
- Resource-Allocation Graph Scheme
  - Example: Safe State



safe sequence: < P 1, P 2 >

Deadlock Avoidance (死锁避免)

- Resource-Allocation Graph Scheme
  - Example: Unsafe State in Resource-Allocation Graph



Unsafe State In Resource-Allocation Graph

Deadlock Avoidance (死锁避免)

- Resource-Allocation Graph Scheme
  - Resource-Allocation Graph Algorithm
    - ✓ Suppose that process Pi requests a resource Rj
    - ✓ 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

Deadlock Avoidance (死锁避免)

- Banker's Algorithm (银行家算法)
  - Multiple instances
  - Each process must have 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
- Safety algorithm
- Resource-request algorithm

Deadlock Avoidance (死锁避免)

- Data structures
  - 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 Rj available
    - ✓ Max: n × m matrix. If Max [i, j] = k, then process Pi may request at most k instances of resource type Rj
    - ✓ Allocation: n × m matrix. If Allocation[i, j] = k then Pi is currently allocated k instances of Rj
    - ✓ Need: n × m matrix. If Need[i, j] = k, then Pi may need k more instances of Rj to complete its task
    - ✓ Need [i, j] = Max[i, j] – Allocation [i, j]

Deadlock Avoidance (死锁避免)

- Safety Algorithm
  - 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 an 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

Deadlock Avoidance (死锁避免)

- Resource-Request Algorithm for Process Pi

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 $Request_i \le Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $Request_i \le Available$, 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:

$$Available = Available - Request;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

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

Deadlock Avoidance (死锁避免)

- **Example of Banker's Algorithm**
- *5 processes P0 through P4;*
- *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
P0	0 1 0	7 5 3	3 3 2
P1	2 0 0	3 2 2	
P2	3 0 2	9 0 2	
P3	2 1 1	2 2 2	
P4	0 0 2	4 3 3	

Deadlock Avoidance (死锁避免)

- Example of Banker’s Algorithm
- *The content of the matrix Need is defined to be Max – Allocation*

	Need
	A B C
P0	7 4 3
P1	1 2 2
P2	6 0 0
P3	0 1 1
P4	4 3 1

- *The system is in a safe state since the sequence < P1, P3, P4, P2, P0> satisfies safety criteria*

Deadlock Avoidance (死锁避免)

- Example: P1 Request (1,0,2)

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

	Allocation	*Need*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	7 4 3	3 3 2 → 2 3 0
P_1	2 0 0 → 3 0 2	1 2 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 $< P_1, P_3, P_4, P_0, P_2>$ satisfies safety requirement
- Can request for (3,3,0) by P_4 be granted?
- Can request for (0,2,0) by P_0 be granted?

Catalog Description

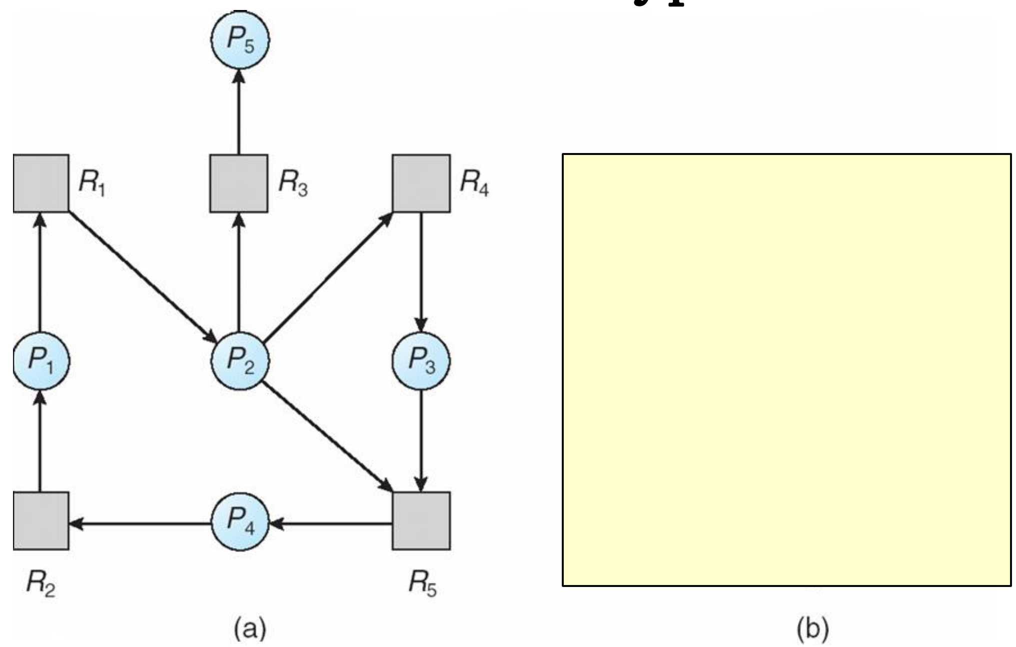
- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Deadlock Detection (死锁检测)

- Allow system to enter deadlock state
  - Detection algorithm
    - ✓ single instance
    - ✓ several instances
  - Recovery scheme
    - ✓ Process termination
    - ✓ Resource preemption

Deadlock Detection (死锁检测)

- Single Instance of Each Resource Type
- Maintain wait-for graph
  - Nodes are processes
  - Pi → Pj if Pi is waiting for Pj



Resource-Allocation Graph Corresponding wait-for graph

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

Deadlock Detection (死锁检测)

- Several Instances of a Resource Type
  - Data structures:
    - ✓ 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 Pi is requesting k more instances of resource type Rj.

Deadlock Detection (死锁检测)

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:

   (a) *Work* = *Available*

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

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

Deadlock Detection (死锁检测)

Example of Detection Algorithm

- Five processes P_0 - 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*

Deadlock Detection (死锁检测)

Example of Detection Algorithm

- *If* 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 0 → 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

Deadlock Detection (死锁检测)

- Example of Detection Algorithm
  - ✓ When, and how often, to invoke the algorithm, depends on:
  - How often is a deadlock likely to occur?
  - How many processes will need to be rolled back?

  - ✓ If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would generally not be able to tell which of the many deadlocked processes "caused" the deadlock.

Catalog Description

- The Deadlock Problem
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Recovery from Deadlock

- Process Termination
  - ✓ Abort all deadlocked processes
  - ✓ Abort one process at a time until the deadlock cycle is eliminated
  - ✓ To minimize cost: In which order should we choose to abort?
    - –Priority of the process
    - –How long the process has computed, and how much longer the process will compute before completing its designated task
    - –Resources the process has used
    - –Resources the process needs to complete
    - –How many processes will need to be terminated
    - –Is the process interactive or batch?

Recovery from Deadlock

- Resource Preemption
  - Three issues need to be addressed:
    - ✓ Selecting a victim – minimize cost
    - ✓ Rollback – return to some safe state, restart the process from that state
    - ✓ Starvation – the same process may always be picked as a victim; the most common solution is to include the number of rollbacks in the cost factor.

End of Chapter 5