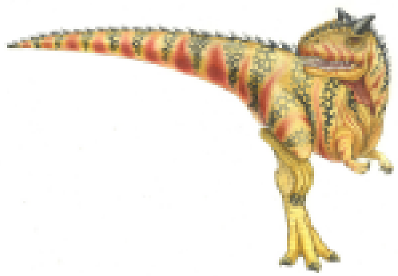
Chapter 8: Deadlocks



Outline

- System Model
- Deadlock Characterization

- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock

Chapter Objectives

- Illustrate how deadlock can occur when mutex locks areused
- Define the four

necessary conditions that characterize deadlock▪ Identify a deadlock situation in a resource allocation graph▪ Evaluate the four different approaches for preventing deadlocks▪ Apply the banker's algorithm for deadlock avoidance ▪ Apply the deadlock detection algorithm ▪ Evaluate approaches for recovering from deadlock

SystemModel

- System consists of resources

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

Deadlock withSemaphores

- Data:
  - A semaphore **S1** initialized to 1
  - A semaphore **S2** initialized to 1
- Two processes P1 and P2
- **P1:**

```
wait(s1)
wait(s2)
```

- **P2:**

```
wait(s2)
wait(s1)
```

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **Mutual exclusion**: only one process at a time can usearesource
- **Hold and wait**: a process holding at least one resourceiswaiting to acquire additional resources held by other processes
- **No preemption**: a resource can be released only voluntarilyby the process holding it, after that process has completeditstask
- **Circular wait**: there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that isheldby P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waitingfor a resource that is held by P_0.

Resource-AllocationGraph

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

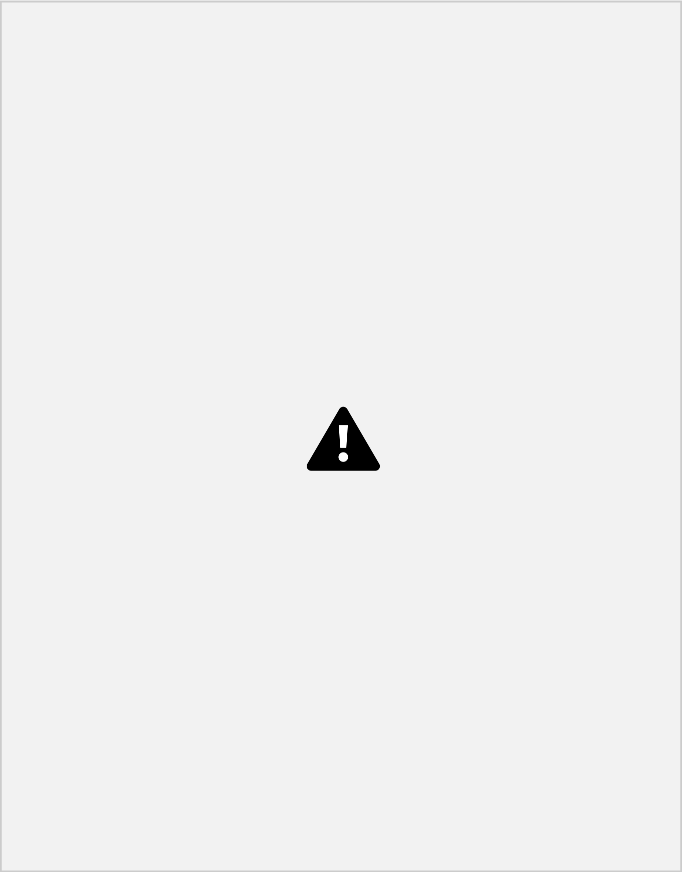
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed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

Resource AllocationGraphExample

- One instance of R1
- Two instances of R2
- One instance of R3
- Three instance of R4
- T1 holds one instance of R2 and is waiting for an instance of R1
- T2 holds one instance of R1, one instance of R2, and is waiting for an instance of R3
- T3 is holds one instance of R3

Resource Allocation GraphwithaDeadlock

Graph with a CycleBut noDeadlock



Basic Facts

- If graph contains no cycles ⇒ no deadlock
- If graph contains a cycle ⇒
  - If only one instance per resource type, then deadlock
  - If several instances per resource type, possibility of deadlock

Methods for HandlingDeadlocks

- 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 ▪ Ignore the problem and pretend that deadlocks never occur inthesystem.

Deadlock Prevention

Invalidate one of the four necessary conditions for deadlock:

- **Mutual Exclusion** – not required for sharable resources(e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a processrequests a resource, it does not hold any other resources
  - Require process to request and be allocated all itsresourcesbefore it begins execution, or allow process to request resources only when the process has none allocatedtoit.
  - Low resource utilization; starvation possible

Deadlock Prevention(Cont.)

▪ **No Preemption**: (stealing resources mid-way)

- If a process that is holding some resources requests another resourcethatcannot be immediately allocated to it, then all resources currentlybeingheld are released.
- Preempted resources are added to the list of resources for whichtheprocess is waiting. Process will be restarted only whenit canregainitsoldresources, as well as the new ones that it is requesting

 Eg1: P1 preempts CD ROM and Hard drive if printer is unavailable. Eg2: If P2 has printer and is waiting for Hard Disk drive, it cbepreemptedand Printer can be given to P1.

 When aprocess (P1) request resources, we check whether theyareavailable or not. If they are available we allocate themelsewecheckthatwhether they are allocated to some other waiting processr (P2). If sowepreempt the resources from the waiting process(P2) andallocatethemtothe requesting process (PI). If resource is not availablewithP2, PI hasto wait.

Circular Wait

- **Circular Wait:**
  - Impose a total ordering of all resource types, and requirethat eachprocessrequests resources in an increasing order of enumeration
- Invalidating the circular wait condition is most common.
- Simply assign each resource (i.e., mutex locks) a unique number.
- Resources must be acquired in order.
- If: **`CD ROM drive = 1, Hard disk drive = 5, Printer=12`**
- **`A process holding 'disk drive' , cannot requestforCDROM drive, but it can request for printer.`**
- **`Eg: semaphore`**

`P1: P2:`

```
wait(s1) wait(s2) s1 = 1wait(s2) wait(s1) s2 = 5
```


Circular Wait

- Deadlock can be prevented by using the following protccol for circularwait:
- Each process can request the resource in increasing order. Aprocesscan request any number of instances of resource type say Ri initiallyandthenitcanrequest instances of resource type Rj oniy if F(Rj) >F(Ri).
- Alternatively, a process can request for an instance of resourceof lowernumberby releasig all the higher number resources.

Deadlock Avoidance

Requires that the system has some additional ***a priori*** informationavailable

- Simplest and most useful model requires that eachprocess declare the ***maximum number*** of resources of eachtypethat it may need

- The deadlock-avoidance algorithm dynamically examinesthe resource-allocation state to ensure that there cannever be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximumdemands of the processes

Safe State

- When a process requests an available resource, systemmust decide if immediate allocation leaves the systemin a safestate
- 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

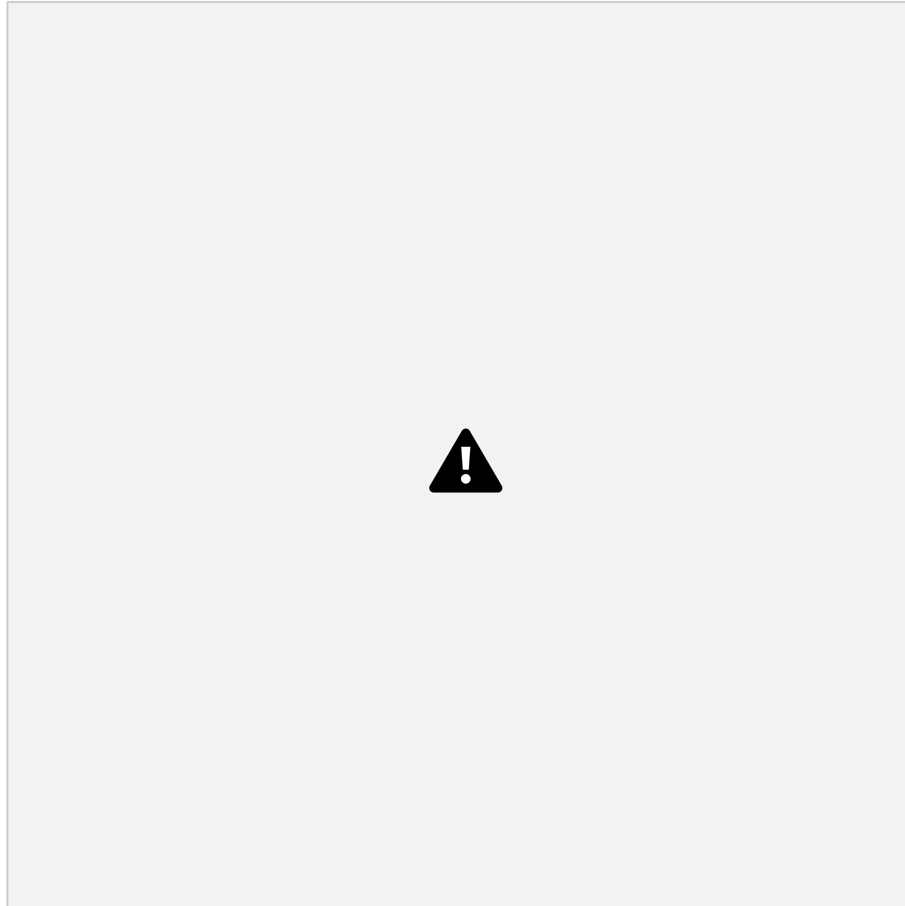
currently available resources + resources held by all the P_j, with $j < I$ ▪ That is:

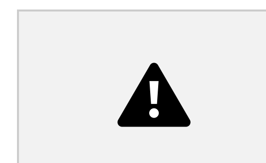
- If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
- When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
- When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Basic Facts

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

Safe, Unsafe, DeadlockState





Avoidance Algorithms

- Single instance of a resource type
  - Use a modified resource-allocation graph
- Multiple instances of a resource type
  - Use the Banker’s Algorithm

Modified Resource-AllocationGraphScheme

- **Claim edge** P_i --> R_j indicates that process P_i may request resource R_j
- **Request edge** $P_i \rightarrow R_j$ indicates that process P_i requests resource R_j
  - Claim edge converts to request edge when a processrequests a resource

- **Assignment edge** $R_j \rightarrow P_i$ indicates that resource R_j was allocated to process P_i
  - Request edge converts to an assignment edge whentheresource is allocated to the process
- When a resource is released by a process, assignment edgereconverts to a claim edge
- Resources must be claimed *a priori* in the system

Resource-AllocationGraph

Unsafe State In

Resource-AllocationGraph

Resource-AllocationGraphAlgorithm

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

Banker's Algorithm

- Multiple instances of resources
- Each process must a priori claim maximumuse
- When a process requests a resource it may have to wait
- When a process gets all its resources it must return themin a finite amount of time

Data Structures for theBanker'sAlgorithm

Let n = number of processes, and m = number of resourcestypes.

- **Available**: Vector of length m. If available [j] = k, therearekinstances of resource type R_j available
- **Max**: n x m matrix. If *Max* [i,j] = k, then process P_i mayrequest at most k instances of resource type R_j
- **Allocation**: n x m matrix. If Allocation[i,j] = k then P_i iscurrently allocated k instances of R_j
- **Need**: n x m matrix. If *Need*[i,j] = k, then P_i may needkmore 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* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:

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

   (b) ***Need$_i$*** ≤ ***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 systemis in a safestate

Resource-Request Algorithm for Process P_i

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

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

Example of Banker 'sAlgorithm

- 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			

Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

<u>*Need*</u>

	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

Example(Cont.)

- Snapshot at time T_0:

Allocation Max Available Need

A B C A B C A B CABC

P_0 0 1 0 7 5 3 3 3 2 7 4 3P_1 2 0 0 3 2 2 1 22P_2 3 0 2 9 0 2 6 00P_3 2 1 1 2 2 2 0 11P_4 0 0 2 4 3 3 4 31

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

Example: P_1 Request (1,0,2)

- Check that Request ≤ Available (that is, (1,0,2) ≤ (3,3,2) ⇒ 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	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
- Can request for (3,3,0) by $\boldsymbol{P_4}$ be granted?
- Can request for (0,2,0) by $\boldsymbol{P_0}$ be granted?

Deadlock Detection

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

Single Instance of EachResourceType

- 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 cycleinthegraph.If there is a cycle, there exists a deadlock

- An algorithm to detect a cycle in a graph requires an order of n^2operations, where n is the number of vertices in the graph

Resource-Allocation Graph andWait-forGraph

Resource-Allocation Graph Corresponding wait-for graph

Several Instancesof aResourceType

- **Available***:* A vector of length ***m*** indicates the number of availableresources of each type
- **Allocation***:* An ***n* x *m*** matrix defines the number of resourcesof eachtype currently allocated to each process
- **Request***:* An ***n* x *m*** matrix indicates the current request of eachprocess.

- If ***Request* [*i*][*j*] = *k***, then process P_i is requesting ***k*** more instances of resource type R_j.

DetectionAlgorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m***and ***n***, respectivelyInitialize:
   a) ***Work* = *Available***
   b) For ***i* = 1,2, …, *n***, if ***Allocation***$_i$ ≠ **0**, then ***Finish*[i] = *false***; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:
   a) ***Finish*[*i*] == *false***
   b) ***Request***$_i$ ≤ ***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 ≤ ***i*** ≤ ***n***, then the systemisindeadlock state.
   Moreover, if ***Finish*[i] == *false***, then ***P_i*** is deadlocked

Algorithm requires an order of O(m x n^2) operations todetect whether the system is in deadlocked state

Example of DetectionAlgorithm

- Five processes P_0through 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***

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

- 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

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: ProcessTermination

- 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 howmuch longer tocompletion
  - 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: ResourcePreemption

- **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

Endof Chapter8

Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

<u>*Need*</u>

	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

Example of Banker 'sAlgorithm

- 5 processes P_0through P_4;

  3 resource types:

  A (10 instances), *B* (5 instances), and *C* (7 instances)
- Snapshot at time T_0:

Allocation Max Available Need

A B C A B C A B C ABC

P_0 0 1 0 7 5 3 3 3 2 7 4 3P_1 2 0 0 3 2 2 1 22P_2 3 0 2 9 0 2 6 00P_3 2 1 1 2 2 2 0 11P_4 0 0 2 4 3 3 4 31

Resource-AllocationGraphScheme

Claim edge $P_i \rightarrow R_j$ indicated that process P_j may request resourceR_j; represented by a dashed line

- Claim edge converts to request edge when a process requestsaresource
- Request edge converted to an assignment edge when theresourceisallocated to the process
- When a resource is released by a process, assignment edgereconverts to a claim edge

- Resources must be claimed *a priori* in the system