Introduction to OS

Deadlock

MOS Ch. 6

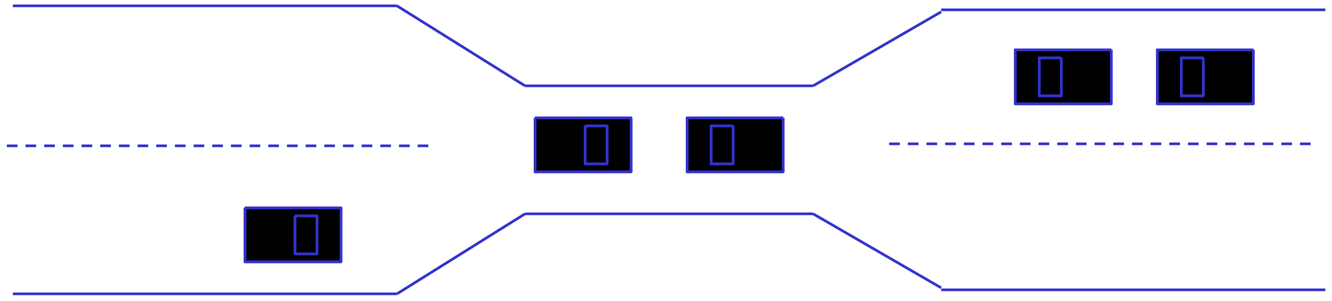
Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Outline

- ***What is Deadlock?***
- How to handle Deadlock?
- Deadlock Detection and Recovery
- Deadlock Avoidance
- Deadlock Prevention

What is Deadlock?



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

System Model

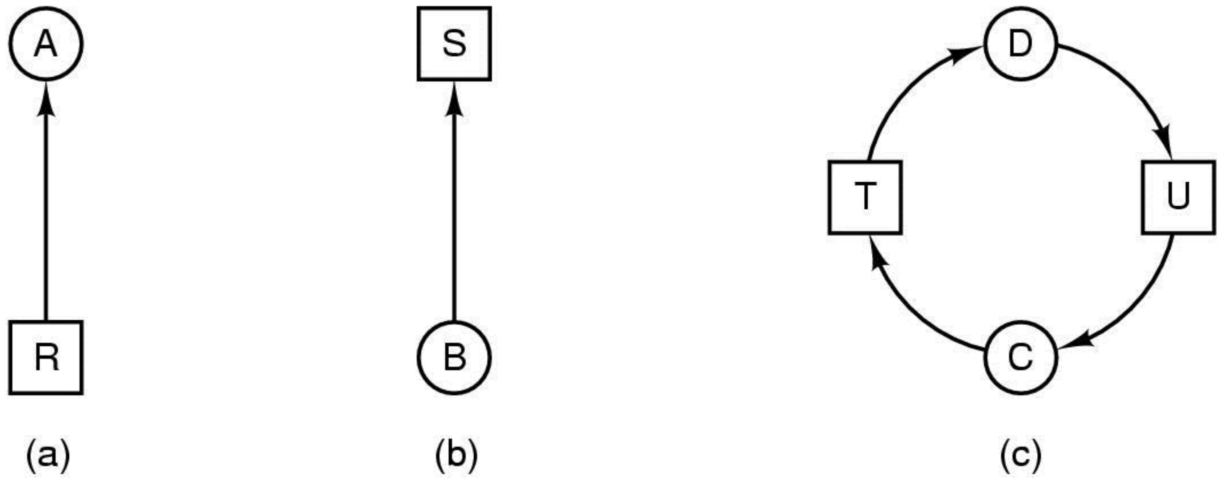
- Resource types: $R_1, R_2, \ldots, R_m$
  - CPU cycles, memory space, I/O devices!
- Each resource type R_i has W_i instances
- Each process utilizes a resource as follows:
  - request
  - use
  - release

Deadlock Characterization

- Deadlock can arise if four conditions hold simultaneously:

1) ***Mutual exclusion:*** only one process at a time can use a resource.
2) ***Hold and wait:*** a process holding resource(s) is waiting to acquire additional resources held by other processes.
3) ***No preemption:*** a resource can be released only voluntarily by the process holding it upon its task completion.
4) ***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 is held by 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 waiting for a resource that is held by P_0.

Resource Allocation Graph



a) resource R assigned to process A

b) process B is requesting/waiting for resource S

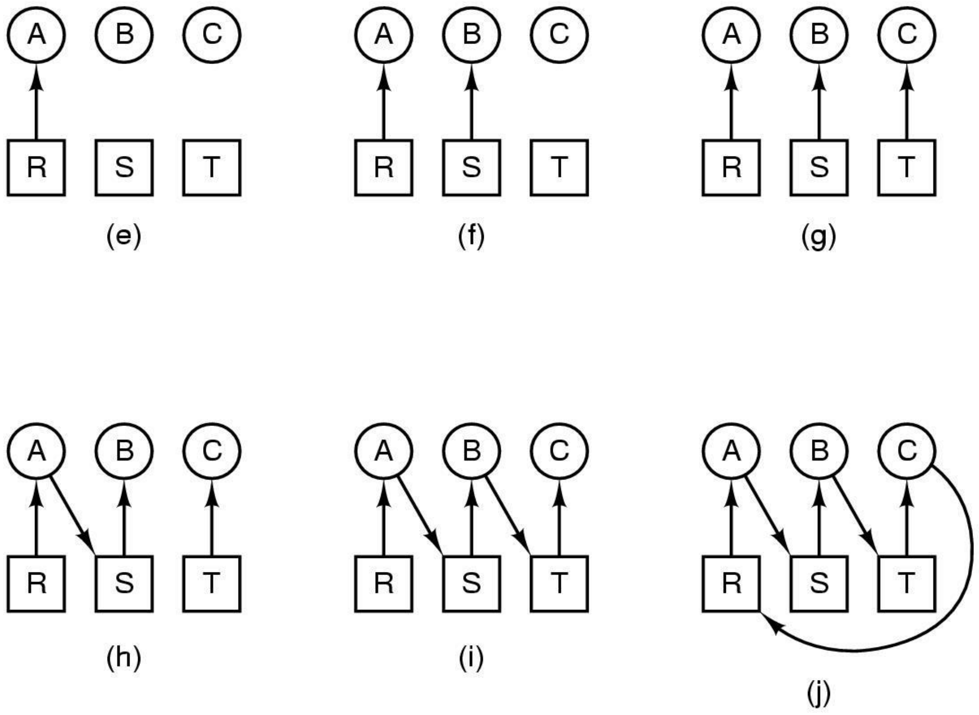
c) process C and D are in deadlock over resources T and U

How Deadlock occurs?

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

(d)



(e) (f) (g)

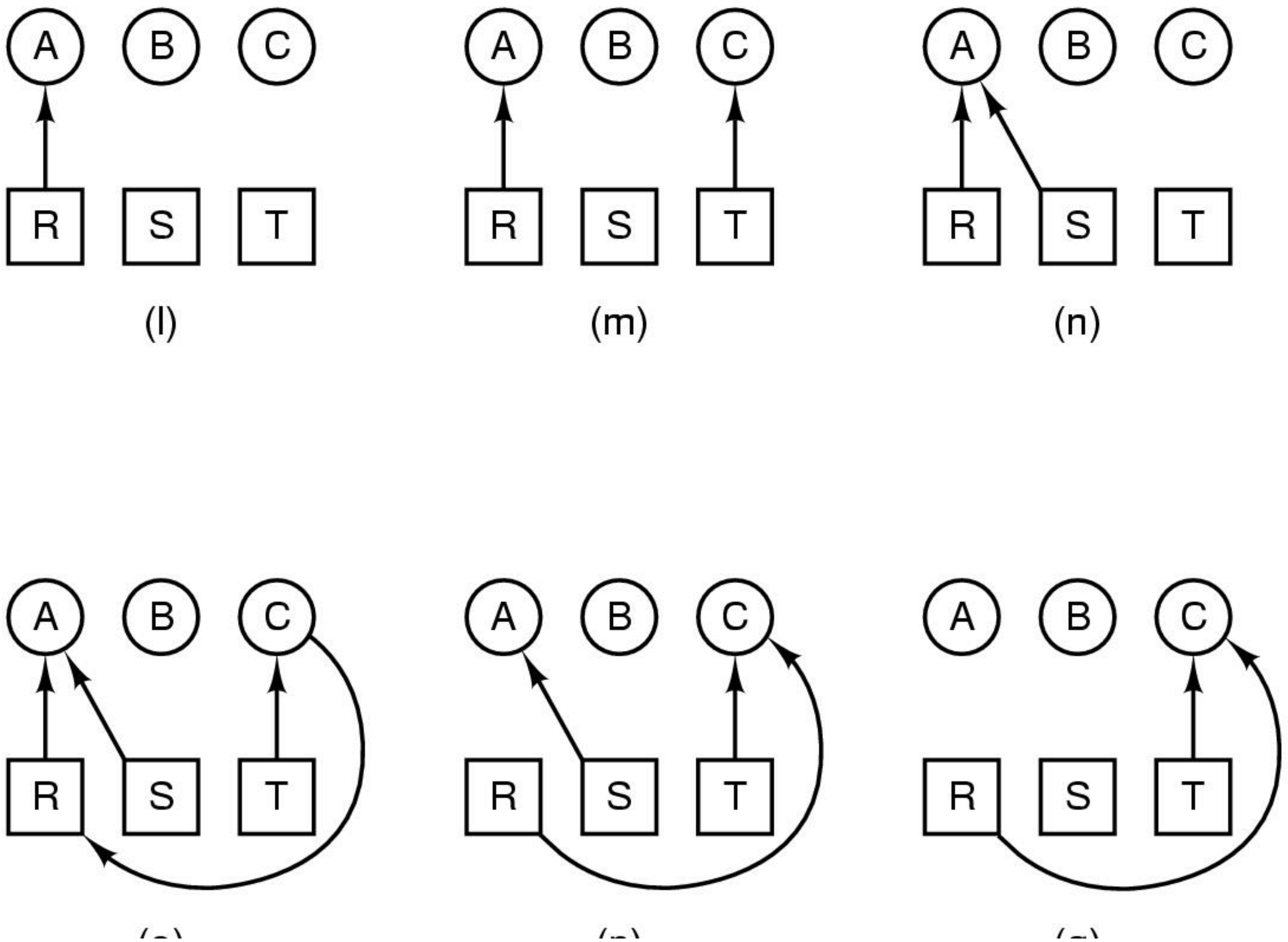
(h) (i) (j)

How Deadlock can be avoided?

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S

   no deadlock

(k)



Outline

- ***What is Deadlock?***
- ***How to handle Deadlock?***
- Deadlock Detection and Recovery
- Deadlock Avoidance
- Deadlock Prevention

Methods for Handling Deadlock

- **Ignore the problem** and pretend that deadlocks never or rarely occur in the system;
  - used by most operating systems, including UNIX.
- Allow the system to enter a deadlock state and then recover. (***Detection and Recovery***)
- Ensure that the system will never enter a deadlock state. (***Prevention and Avoidance***)
  - Avoidance: Careful resource allocation
  - Prevention: Negating one of the four necessary conditions of deadlock

Strategy 1: The Ostrich Algorithm

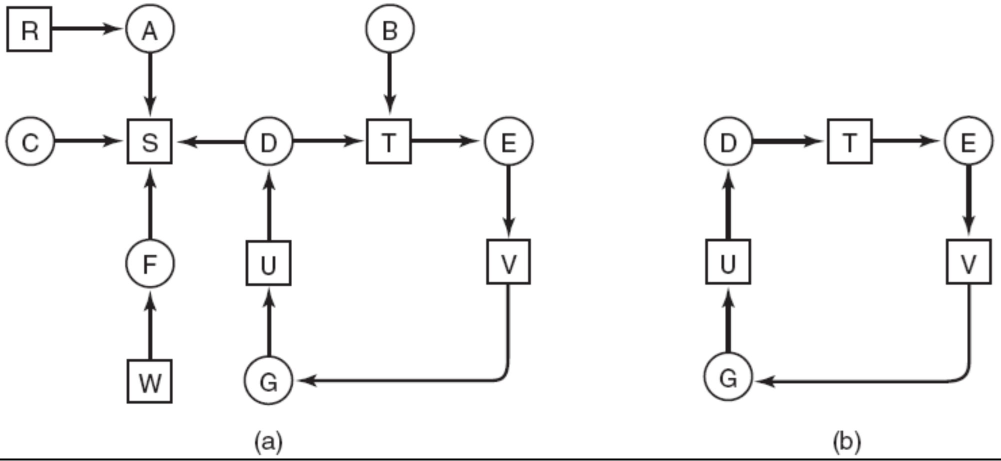
- Just ignore the problem, reasonable if
  - deadlocks occur very rarely
  - cost of prevention is high
- UNIX and Windows takes this approach
- It is a trade off between
  - Performance & convenience
  - correctness

Outline

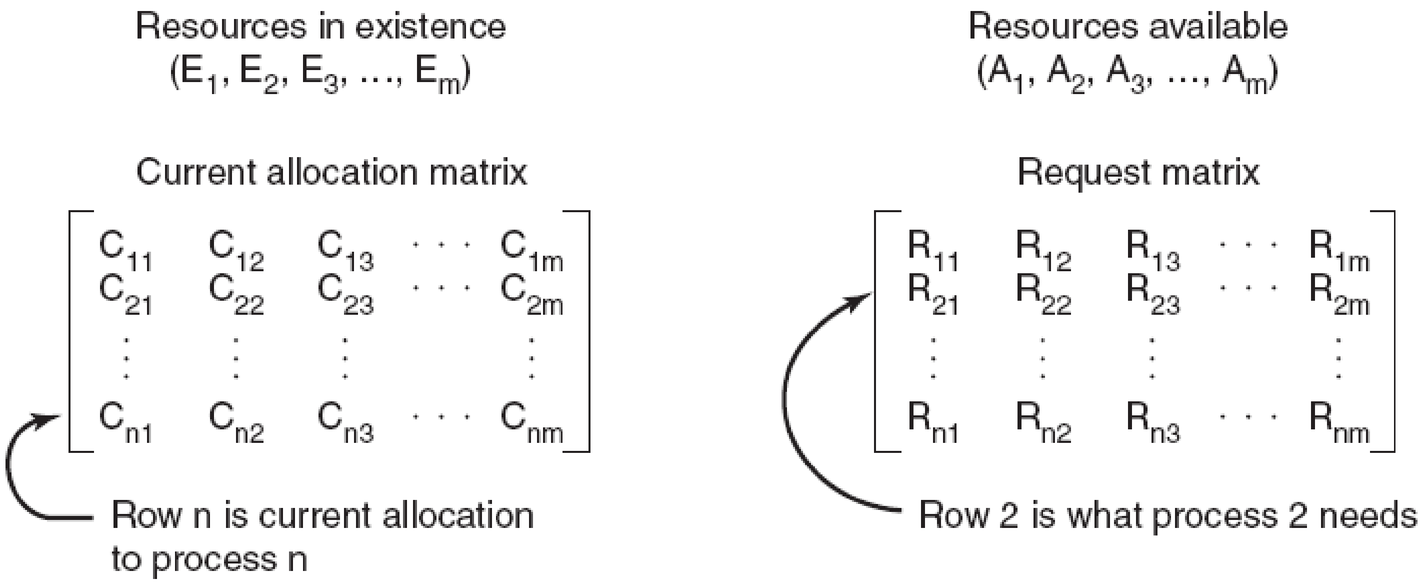
- ***What is Deadlock?***
- ***How to handle Deadlock?***
- ***Deadlock Detection and Recovery***
- Deadlock Avoidance
- Deadlock Prevention

Deadlock Detection with One Resource of Each Type

1. For each node, N in the graph, perform the following five steps with N as the starting node.
2. Initialize L to the empty list, designate all arcs as unmarked.
3. Add current node to end of L, check to see if node now appears in L two times. If it does, graph contains a cycle (listed in L), algorithm terminates.
4. From given node, see if any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick an unmarked outgoing arc at random and mark it. Then follow it to the new current node and go to step 3.
6. If this is initial node, graph does not contain any cycles, algorithm terminates. Otherwise, dead end. Remove it, go back to previous node, make that one current node, go to step 3.



Deadlock Detection with Multiple Resources of Each Type - I


Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$
Resources available
$(A_1, A_2, A_3, \ldots, A_m)$
Current allocation matrix
$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$
Request matrix
$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$
Row n is current allocation to process n
Row 2 is what process 2 needs

- i denote type of resources in E_i and A_i
- E_i and A_i denote the number of resources of type i
- Example, if class 1 is tape drives, then $E_1 = 2$ means the system has two tape drives.

Deadlock Detection with Multiple Resources of Each Type - II

- Each process is initially said to be unmarked. As the algorithm progresses, processes will be marked, indicating that they are able to complete and are thus not deadlocked.

- When the algorithm finishes, all the unmarked processes, if any, are deadlocked.

Deadlock Detection with Multiple Resources of Each Type - III

1. Look for an unmarked process, P_i , for which the i-th row of R is less than or equal to A.
2. If such a process is found, add the i-th row of C to A, mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.

$$\begin{array}{lcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ E = (& 4 & 2 & 3 & 1) \end{array} \qquad \begin{array}{lcccc} & \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ A = (& 2 & 1 & 0 & 0) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Deadlock Detection with Multiple Resources of Each Type - IV

1. Look for an unmarked process, P_i , for which the i-th row of R is less than or equal to A.
2. If such a process is found, add the i-th row of C to A, mark the process, and go back to step 1.
3. If no such process exists, the algorithm terminates.

$$E = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (4 & 2 & 3 & 1) \end{array}$$

$$A = \begin{array}{cccc} \text{Tape drives} & \text{Plotters} & \text{Scanners} & \text{CD Roms} \\ (2 & 1 & 0 & 0) \end{array}$$

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Suppose process 2 needs a CD Rom drive as well

When to Detect Deadlock?

- Every time a resource request is made (very expensive)
- Every k minutes
- or perhaps only when the ***CPU utilization*** has dropped below some threshold !!!

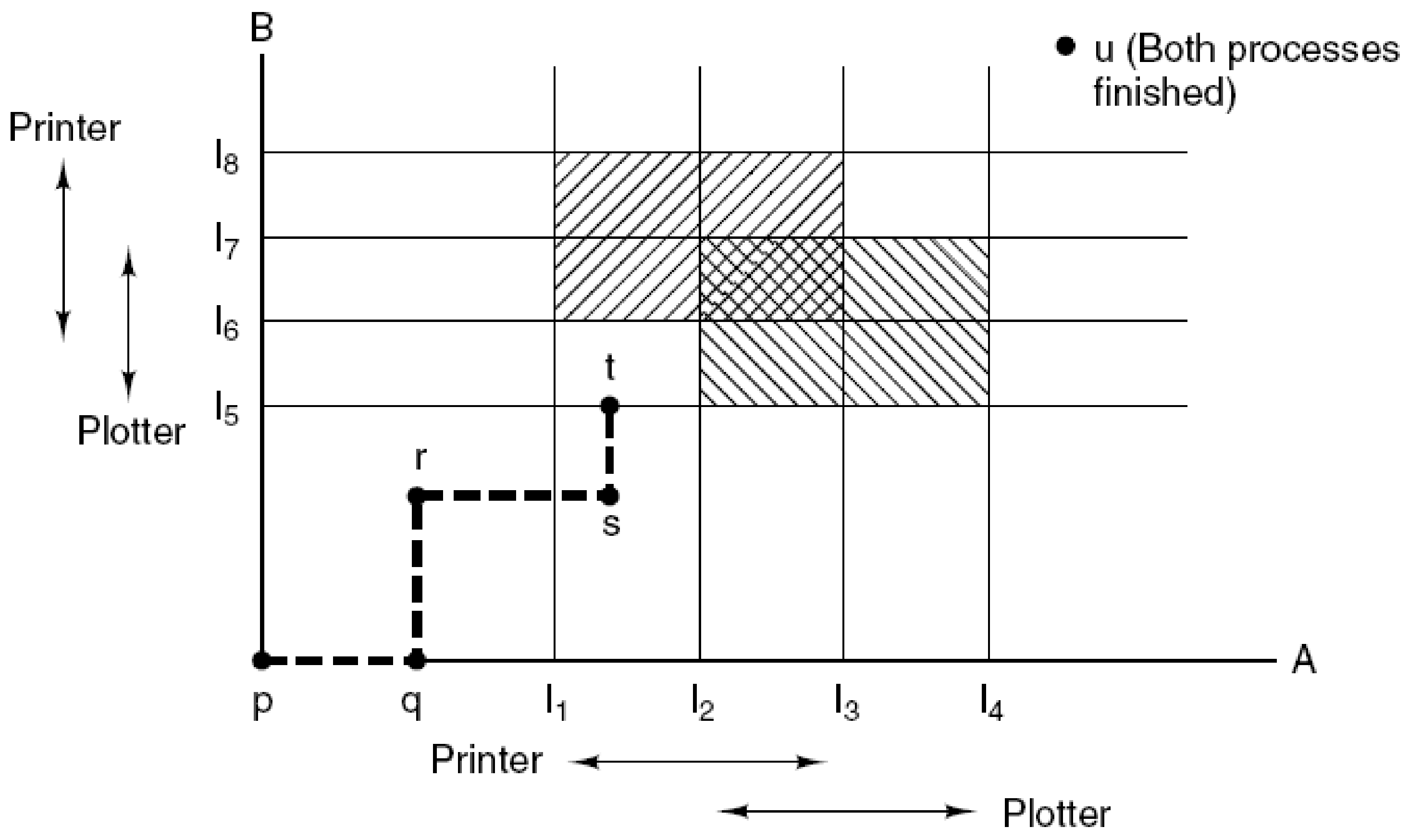
Recovery from Deadlock

- **Recovery through preemption**
  - Example: snatch the printer from the process A, give it to process B. Process B completes. Return the printer to process A. Process A completes.
- **Recovery through rollback**
  - Create a checkpoint for the process A, stop process A, allocate the resource needed to process B. Process B completes. Restart process A from the checkpoint.
- **Recovery through killing processes**
  - Carefully choose a process to kill so that others can proceed.

Outline

- ***What is Deadlock?***
- ***How to handle Deadlock?***
- ***Deadlock Detection and Recovery***
- ***Deadlock Avoidance***
- Deadlock Prevention

Deadlock Avoidance



Two process resource trajectories

Safe State

A state is said to be ***safe*** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1
(b)

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5
(c)

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

Total number of resource's instances=10

Unsafe State

A state is said to be ***safe*** if there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately.

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

$\longrightarrow$ *A* got another resource

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2
(b)

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0
(c)

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4
(d)

Total number of resource's instances=10

Note: An unsafe state is not a deadlocked state. Starting at (b), the system can run for a while. In fact, one process can even complete.

The Banker's Algorithm for a Single Resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

(a)

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

(b)

Safe

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

(c)

B asks for one more unit?

If granting the request leads to an unsafe state, the request is denied, else accepted .

The Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

B is given the printer

E request must be deferred

1. Look for row, R, whose unmet resource needs are all ≤ A. If no such row exists, system will eventually deadlock since no process can run to completion
2. Assume process of row chosen requests all resources it needs and finishes. Mark process as terminated, add all its resources to the A vector.
3. Repeat steps 1 and 2 until either all processes marked terminated (initial state was safe) or no process left whose resource needs can be met (there is a deadlock).

Outline

- ***What is Deadlock?***
- ***How to handle Deadlock?***
- ***Deadlock Detection and Recovery***
- ***Deadlock Avoidance***
- ***Deadlock Prevention***

Deadlock Prevention – I

- **Attacking the mutual exclusion condition**
  - making all relevant resources sharable
  - Some devices (such as printer) can be spooled (shared)
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated
- **Not all devices can be spooled**

Deadlock Prevention – II

- **Attacking the hold and wait condition**
  - requires processes to acquire all their needed resources at once.
  - To acquire new resources in such a system requires a process to give up all the resources it holds and try to reacquire all the resources it needs atomically.
- **processes do not know how many resources they will need until they have started running**
- **Possibility for starvation**

Deadlock Prevention – III

- **Attacking the no preemption condition**
  - Forcibly take away the resource.

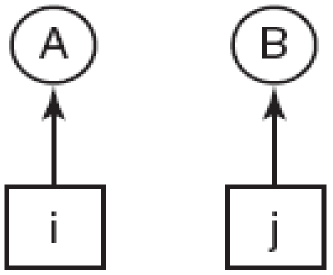
- **Problem, e.g. forcibly taking away the printer while it is printing.**

Deadlock Prevention – IV

- **Attacking the Circular Wait Condition**
  - Processes can request resources whenever they want to, but all requests must be made in numerical order. (avoid cycles)
  - Example: A process may request first a plotter and then a tape drive, but it may not request first a plotter and then a scanner.

1. Imagesetter
2. Scanner
3. Plotter
4. Tape drive
5. CD-ROM drive

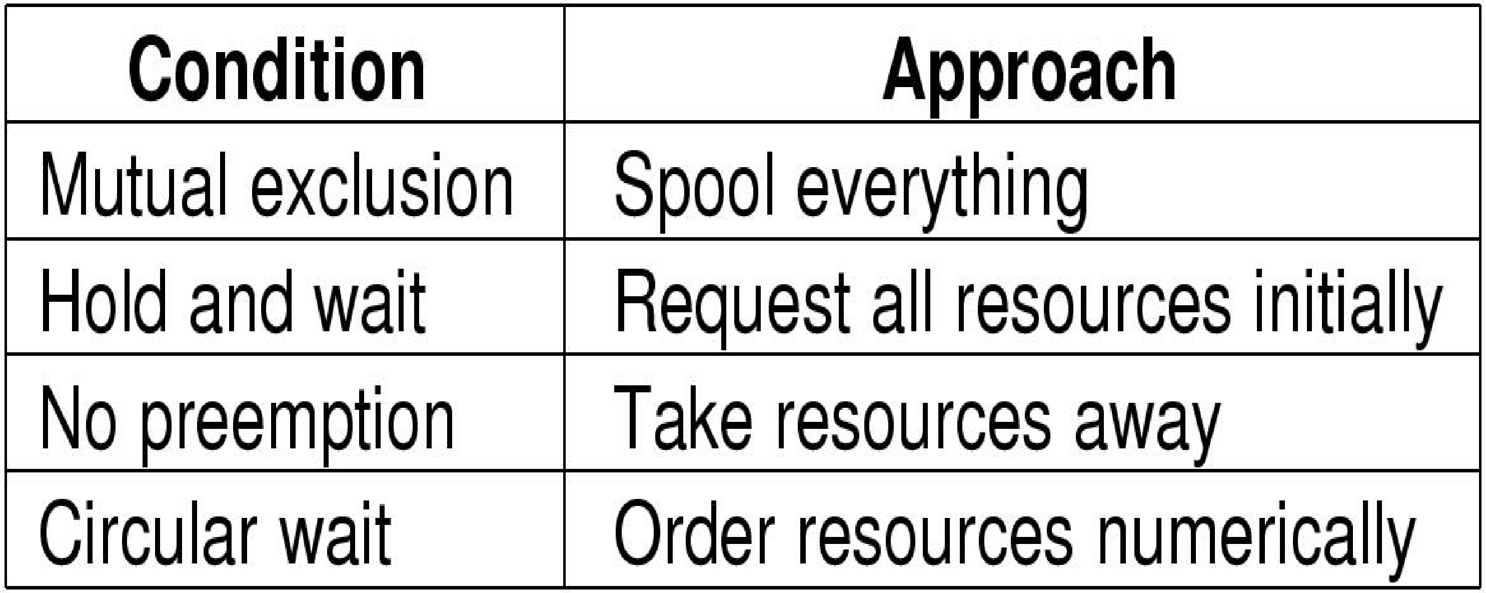
(a)



(b)

- **it may be impossible to find an ordering that satisfies everyone**

Deadlock Prevention Summary



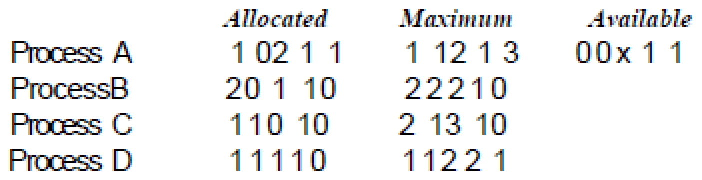
Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Review

- Students working at individual PCs in a computer laboratory send their files to be printed by a server which spools the files on its hard disk. Under what conditions may a deadlock occur if the disk space for the print spool is limited? How may the deadlock be avoided?
- A system has two processes and three identical resources. Each process needs a maximum of two resources. Is deadlock possible? Explain your answer.

Review

- system has four processes and five allocatable resources. The current allocation and maximum needs are as follows:



	Allocated	*Maximum*	*Available*
Process A	1 02 1 1	1 12 1 3	0 0 x 1 1
ProcessB	20 1 10	2 2 2 1 0	
Process C	110 10	2 13 10	
Process D	1 1 1 1 0	1 1 2 2 1	

What is the smallest value of x for which this is a safe state?