Deadlocks

Chapter 6

Preemptable and Nonpreemptable Resources

Sequence of events required to use a resource

1. Request the resource.
2. Use the resource.
3. Release the resource.

Resource Acquisition (1)

```
typedef int semaphore;
semaphore resource_1;


void process_A(void) {
        down(&resource_1);
        use_resource_1( );
        up(&resource_1);
}
```

(a)

```
typedef int semaphore;
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
}
```

(b)

Figure 6-1. Using a semaphore to protect resources. (a) One resource. (b) Two resources.

Resource Acquisition (2)

```
typedef int semaphore;
    semaphore resource_1;
    semaphore resource_2;

    void process_A(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
    }

    void process_B(void) {
        down(&resource_1);
        down(&resource_2);
        use_both_resources( );
        up(&resource_2);
        up(&resource_1);
    }
```

(a)

```
semaphore resource_1;
semaphore resource_2;

void process_A(void) {
    down(&resource_1);
    down(&resource_2);
    use_both_resources( );
    up(&resource_2);
    up(&resource_1);
}

void process_B(void) {
    down(&resource_2);
    down(&resource_1);
    use_both_resources( );
    up(&resource_1);
    up(&resource_2);
}
```

(b)

Figure 6-2. (a) Deadlock-free code.
(b) Code with a potential deadlock.

Deadlock Definition

A set of processes is deadlocked if …

- Each process in the set waiting for an event
- That event can be caused only by another process

Conditions for Resource Deadlocks

Four conditions that must hold:

1. Mutual exclusion
2. Hold and wait
3. No preemption
4. Circular wait condition

Deadlock Modeling (1)

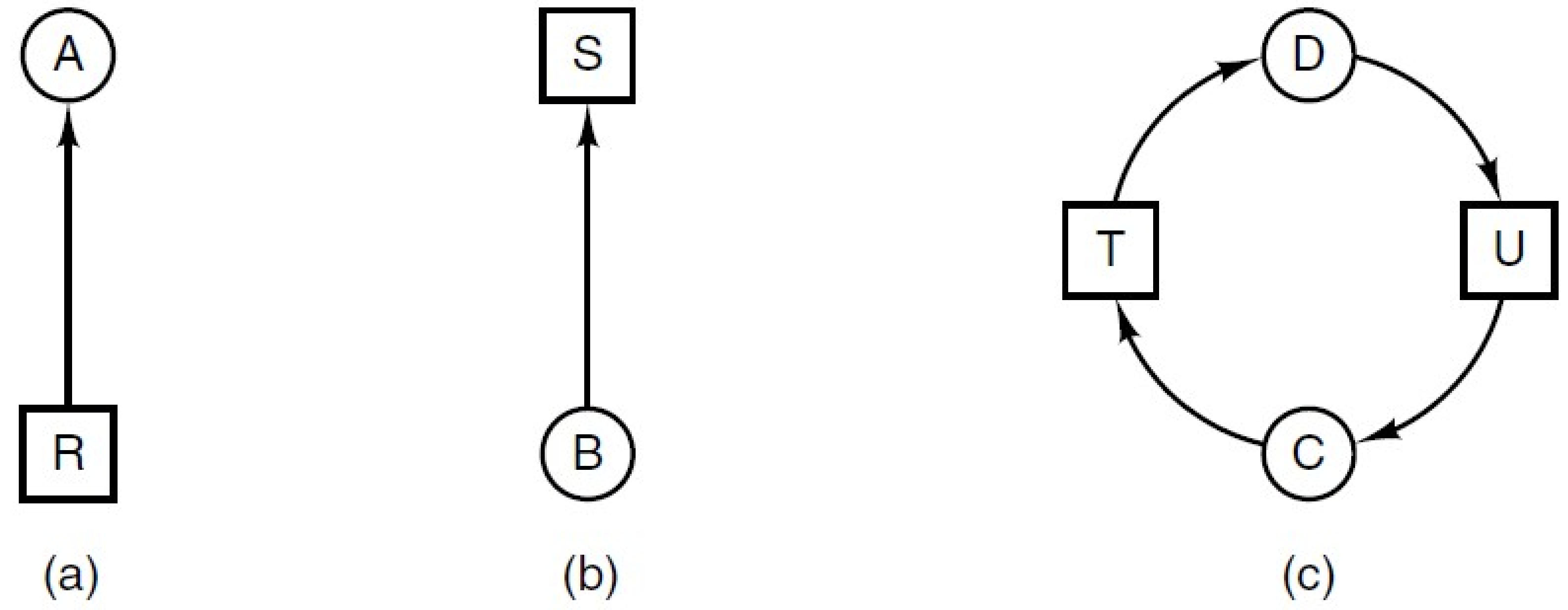
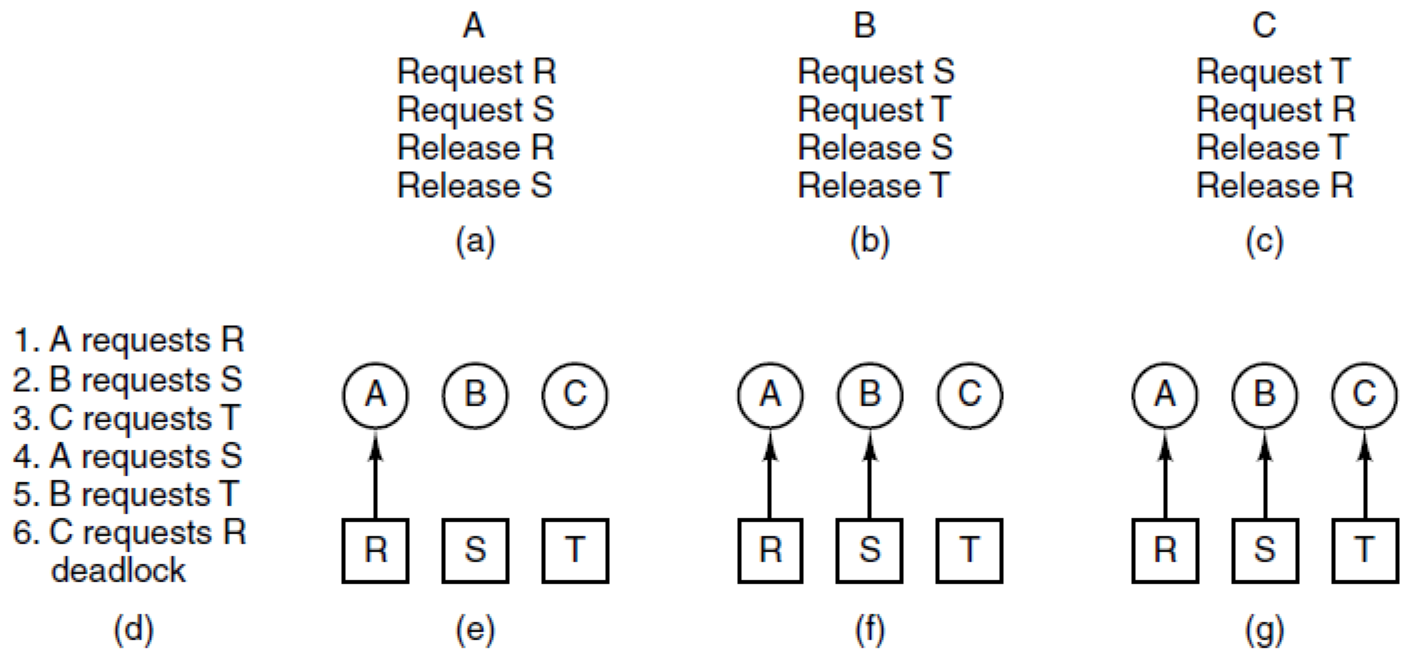


Figure 6-3. Resource allocation graphs. (a) Holding a resource. (b) Requesting a resource. (c) Deadlock.

Deadlock Modeling (2)



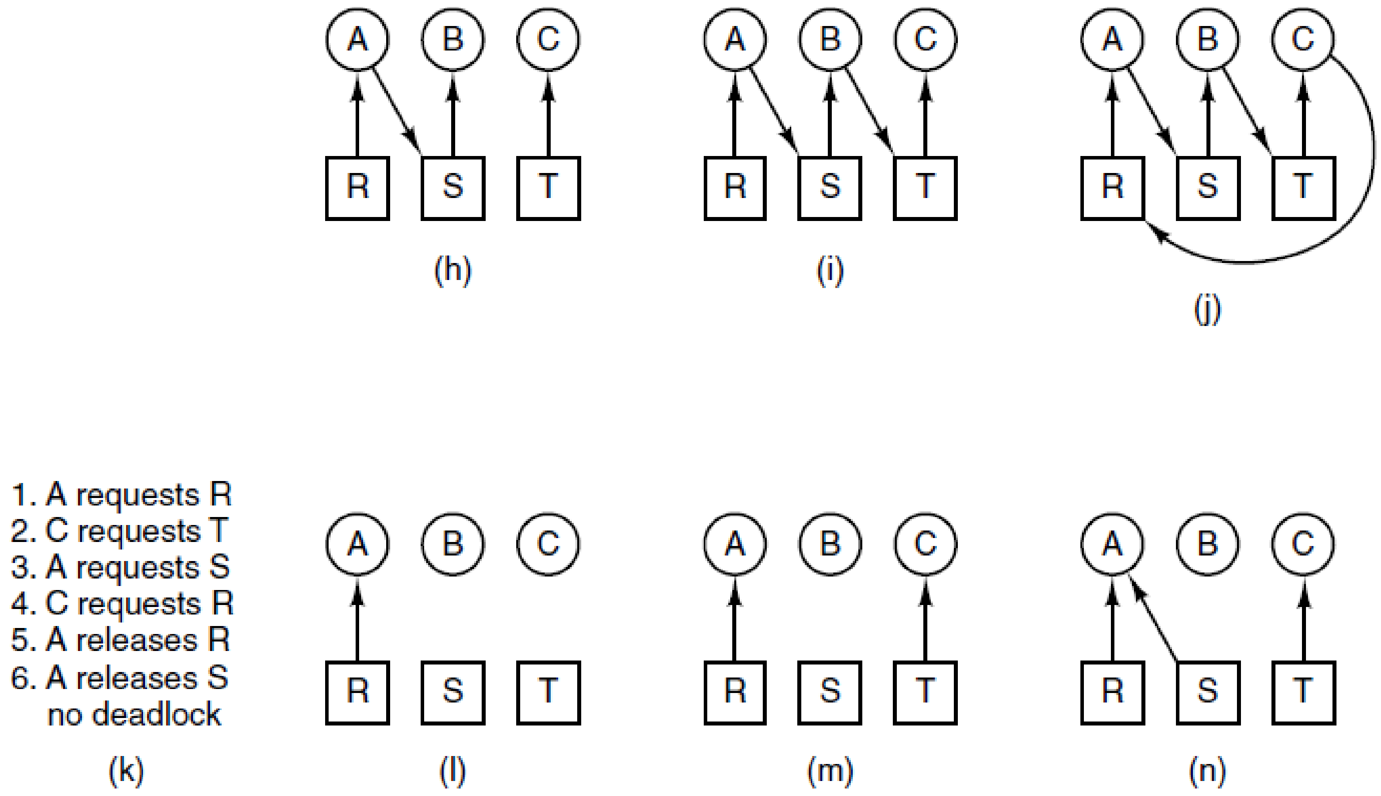
A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R
(a)	(b)	(c)

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)

Figure 6-4. An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling (3)



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

Figure 6-4. An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling (4)

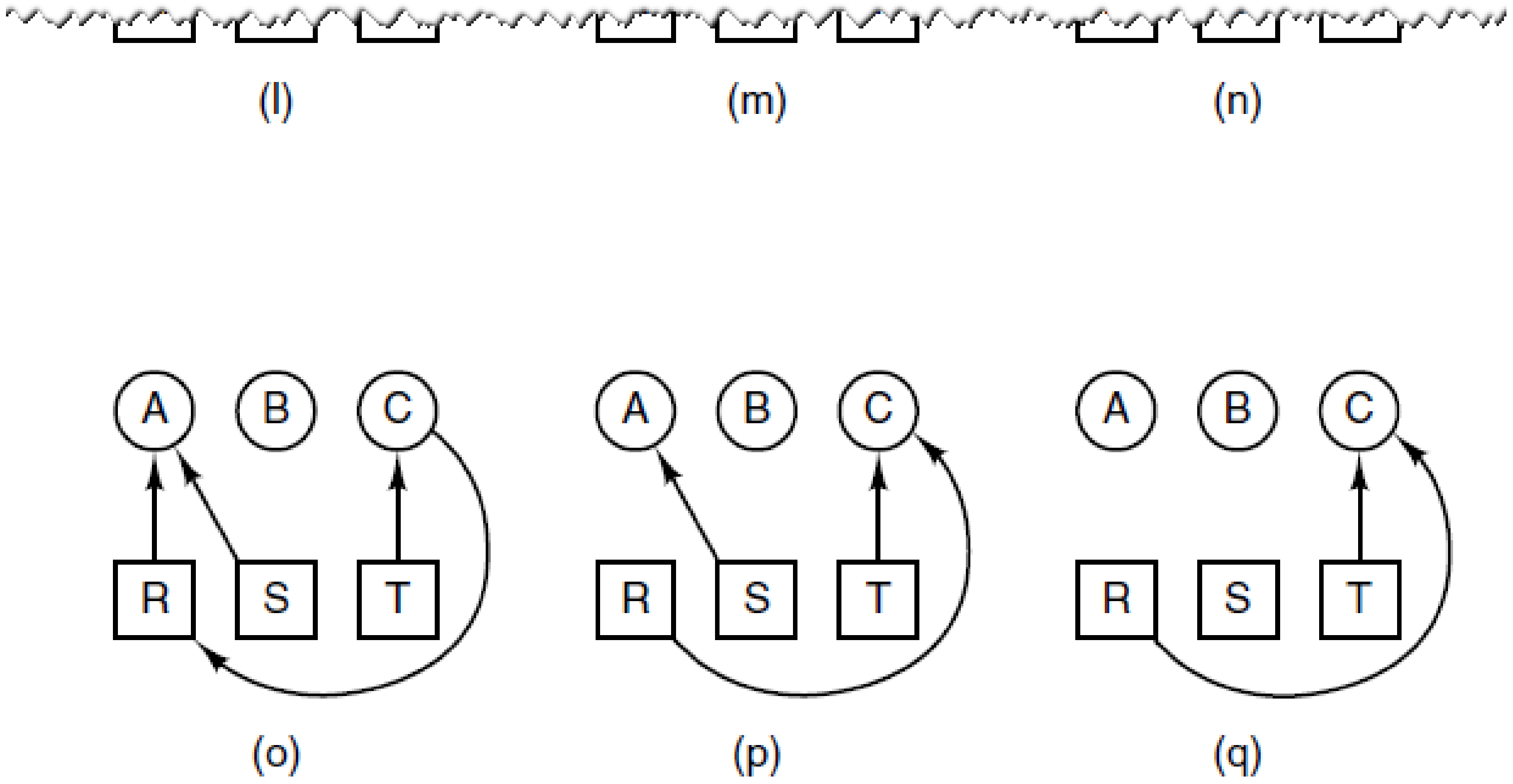


Figure 6-4. An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling (5)

Strategies are used for dealing with deadlocks:

1. Ignore the problem, maybe it will go away.
2. Detection and recovery. Let deadlocks occur, detect them, and take action.
3. Dynamic avoidance by careful resource allocation.
4. Prevention, by structurally negating one of the four required conditions.

Deadlock Detection with One Resource of Each Type (1)

Example of a system – is it deadlocked?

1. Process A holds R, wants S
2. Process B holds nothing, wants T
3. Process C holds nothing, wants S
4. Process D holds U, wants S and T
5. Process E holds T, wants V
6. Process F holds W, wants S
7. Process G holds V, wants U

Deadlock Detection with One Resource of Each Type (2)

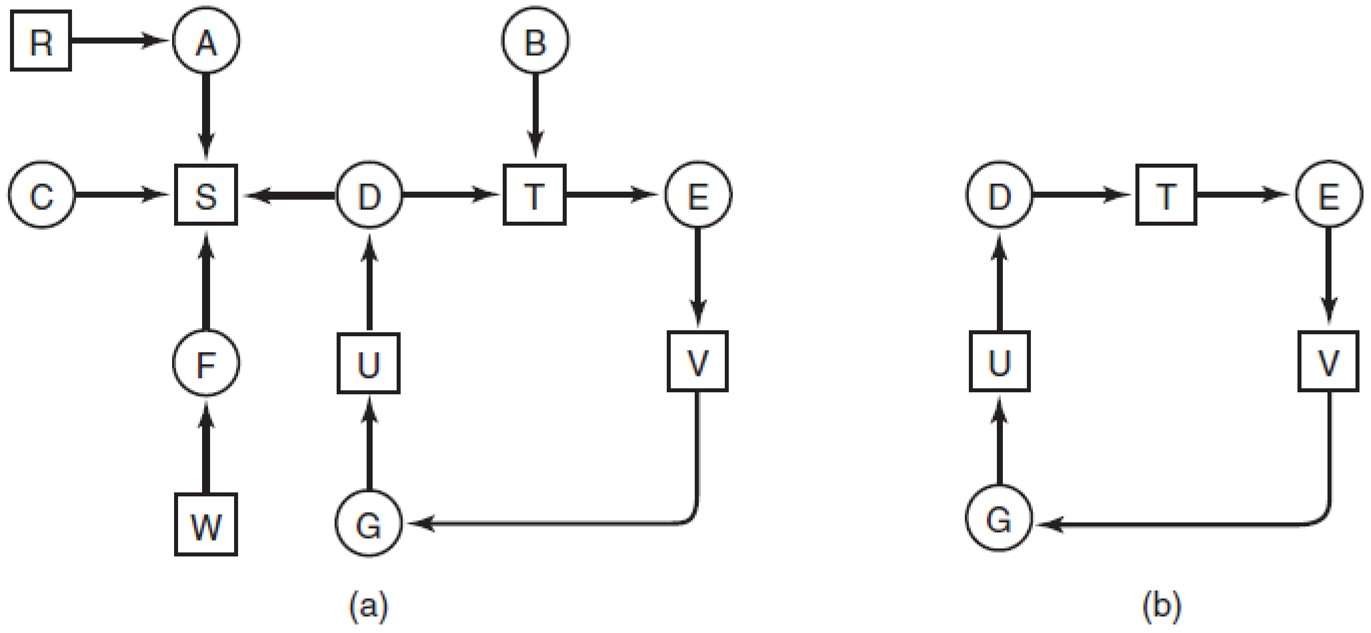


Figure 6-5. (a) A resource graph. (b) A cycle extracted from (a).

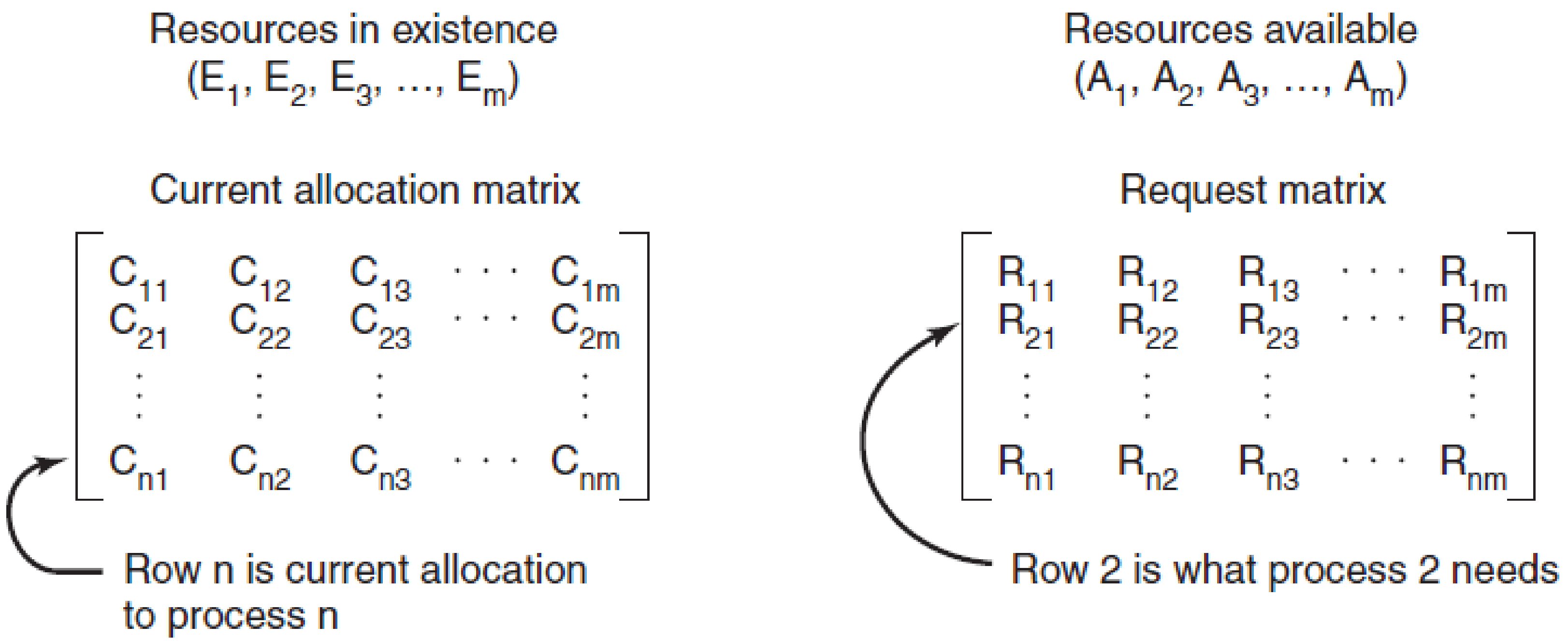
Algorithm to Detect Deadlocks (1)

1. For each node, *N* in the graph, perform following five steps with *N* as starting node.
2. Initialize *L* to empty list, and 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 so, graph contains a cycle (listed in L) and algorithm terminates

Algorithm to Detect Deadlocks (2)

4. From given node, see if there are any unmarked outgoing arcs. If so, go to step 5; if not, go to step 6.
5. Pick unmarked outgoing arc at random, mark it. Then follow to new current node and go to step 3.
6. If this is initial node, graph does not contain cycles, algorithm terminates. Otherwise, dead end. Remove it and go back to the previous node.

Deadlock Detection with Multiple Resources of Each Type (1)


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
Request matrix
Row n is current allocation to process n
Row 2 is what process 2 needs

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix} \qquad \begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Figure 6-6. The four data structures needed by the deadlock detection algorithm.

Deadlock Detection with Multiple Resources of Each Type (2)

Deadlock detection algorithm:

1.Look for 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, go back to step 1.

3.If no such process exists, algorithm terminates.

Deadlock Detection with Multiple Resources of Each Type (3)

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

Current allocation matrix

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

Request matrix

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

Figure 6-7. An example for the deadlock detection algorithm.

Recovery from Deadlock

Possible Methods of recovery (though none are “attractive”):

1. Preemption
2. Rollback
3. Killing processes

Deadlock Avoidance
Resource Trajectories

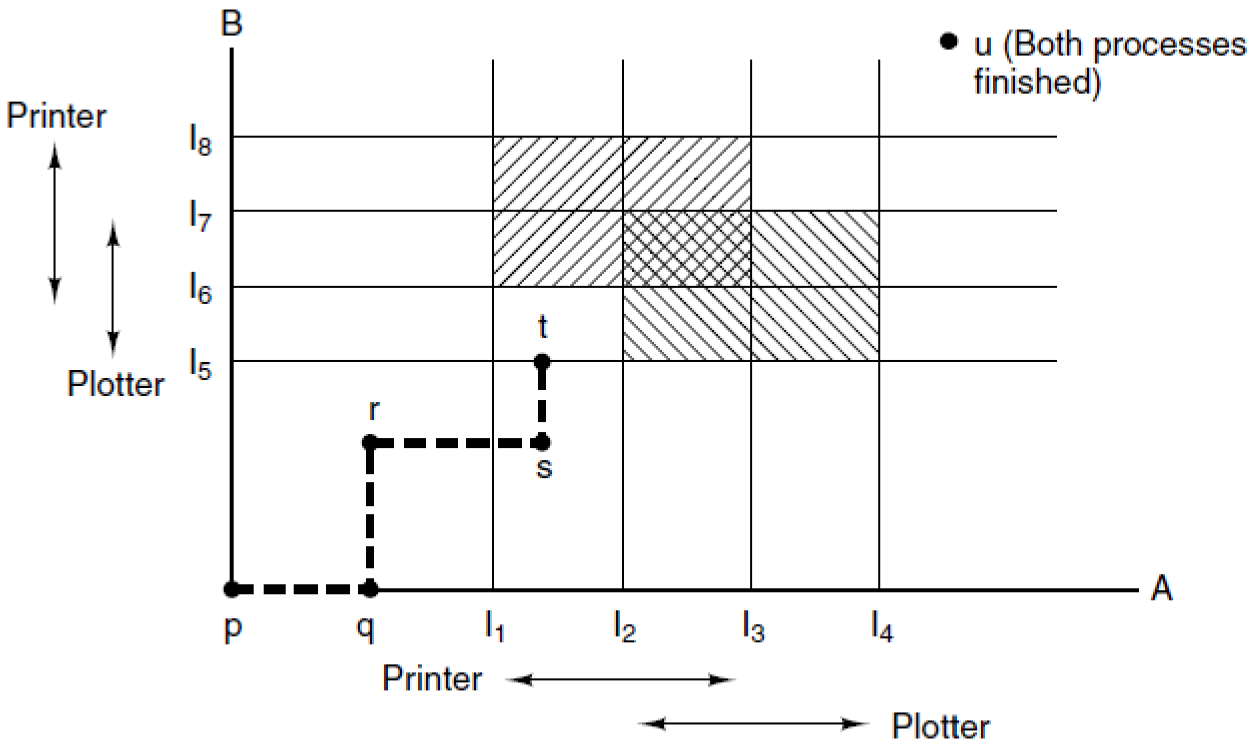


Figure 6-8. Two process resource trajectories.

Safe and Unsafe States (1)

	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)

Figure 6-9. Demonstration that the state in (a) is safe.

Safe and Unsafe States (2)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

	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)

Figure 6-10. Demonstration that the state in (b) is not safe.

Banker's Algorithm for 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)

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

Free: 1

(c)

Figure 6-11. Three resource allocation states:
(a) Safe. (b) Safe. (c) Unsafe.

Banker's Algorithm for Multiple Resources (1)

Resources assigned

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 still needed

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

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

Figure 6-12. The banker's algorithm with multiple resources.

Banker's Algorithm for Multiple Resources (2)

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

Deadlock Prevention

Assure that at least one of conditions is never satisfied

- Mutual exclusion
- Hold and wait
- No Preemption
- Circular wait

Attacking Circular Wait Condition (1)

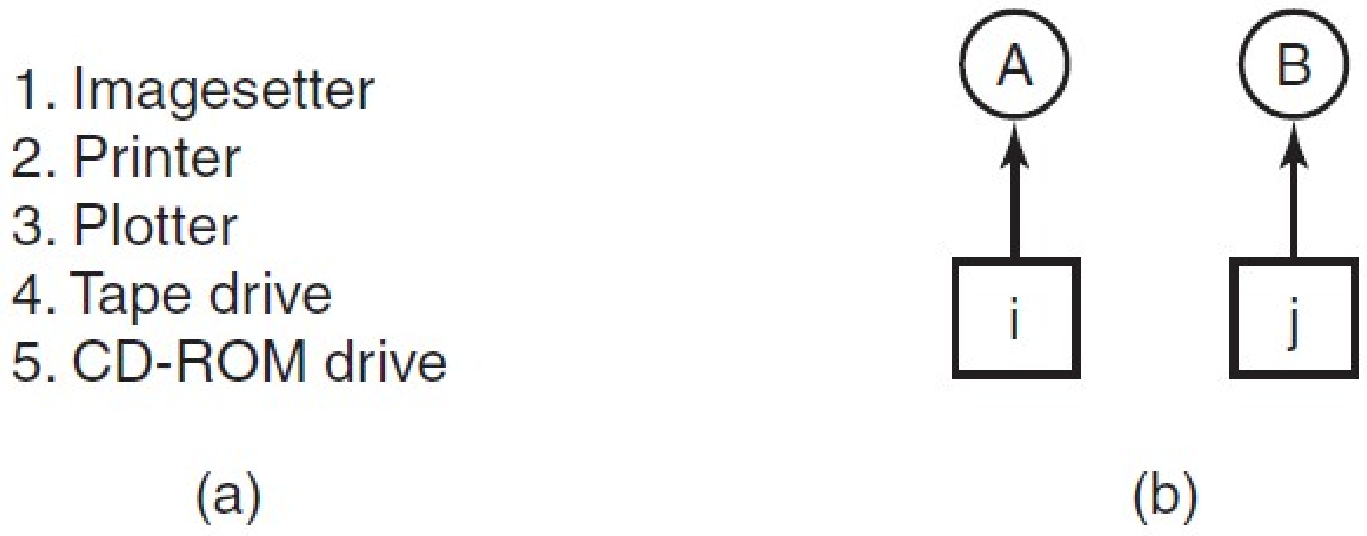

1. Imagesetter
2. Printer
3. Plotter
4. Tape drive
5. CD-ROM drive

Figure 6-13. (a) Numerically ordered resources.
(b) A resource graph

Attacking Circular Wait Condition (2)

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

Figure 6-14. Summary of approaches to deadlock prevention.

Communication Deadlocks

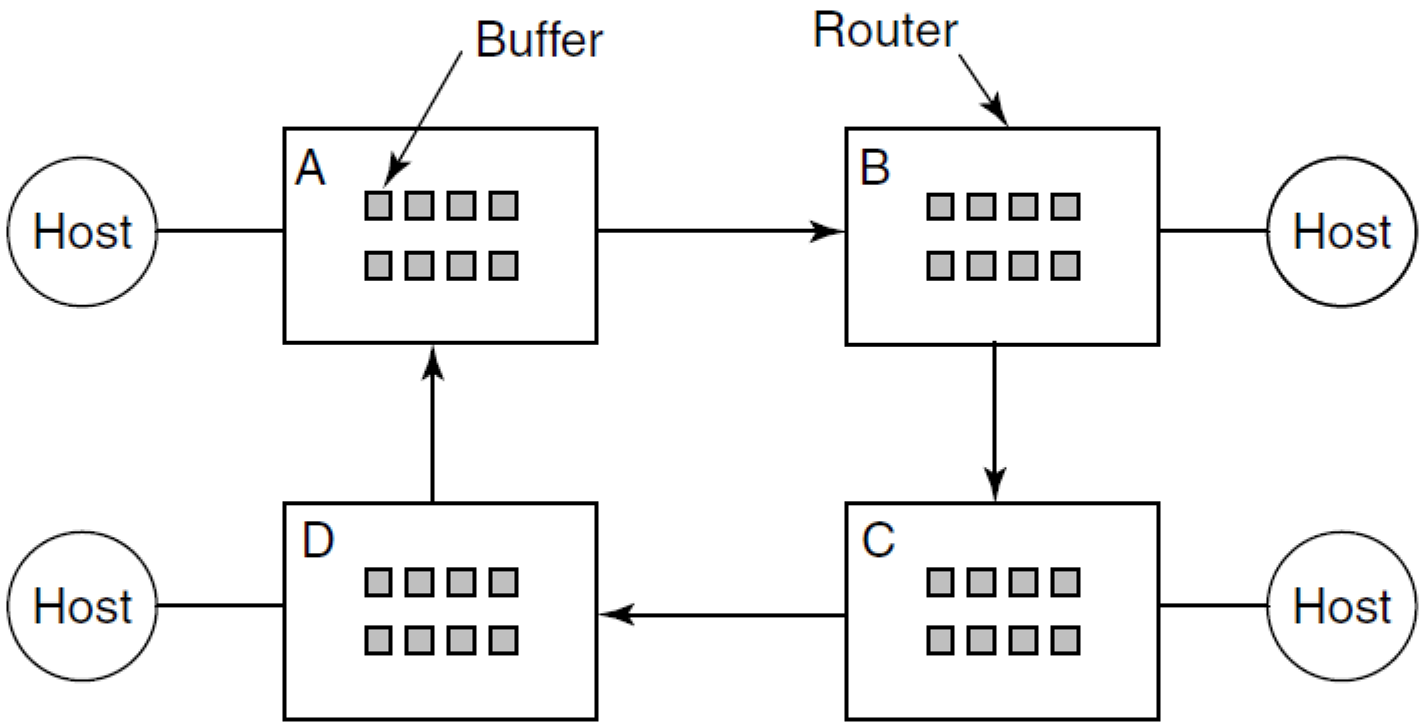


Figure 6-15. A resource deadlock in a network.

Livelock

```
void process_A(void) {
        enter_region(&resource_1);
        enter_region(&resource_2);
        use_both_resources( );
        leave_region(&resource_2);
        leave_region(&resource_1);
}
```

```
void process_B(void) {
        enter_region(&resource_2);
        enter_region(&resource_1);
        use_both_resources( );
        leave_region(&resource_1);
        leave_region(&resource_2);
}
```

Figure 6-16. Busy waiting that can lead to livelock.

End

Chapter 6