MODERN OPERATING SYSTEMS

Chapter 3 Deadlocks

Resource
Introduction to deadlocks
The ostrich algorithm
Deadlock detection and recovery
Deadlock avoidance
Deadlock prevention

Resources

- Examples of computer resources
  - printers
  - tape drives
  - tables

- Processes need access to resources in reasonable order

- Suppose a process holds resource A and requests resource B
  - at same time another process holds B and requests A
  - both are blocked and remain so

Preemptable and Nonpreemptable Resources (可抢占/不可抢占资源)

- Deadlocks occur when …
  - processes are granted exclusive access to devices (排他的访问)
  - we refer to these devices generally as **resources**

- Preemptable resources
  - can be taken away from a process with no ill effects

- Nonpreemptable resources
  - will cause the process to fail if taken away

Resources

- **Sequence** of events required to use a resource
  1. **request** the resource
  2. **use** the resource
  3. **release** the resource

- Must **wait** if request is denied
  - requesting process may be **blocked**
  - may **fail** with error code

Resource Acquisition (资源获取)

```
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)

Using a **semaphore** to protect resources.
(a) One resource. (b) Two resources.

Deadlock-free code

(a) Deadlock-free code.

```
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)

Code with a potential deadlock

(b) Code with a potential deadlock

```
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)

Introduction to Deadlocks

- Formal definition :
  A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.

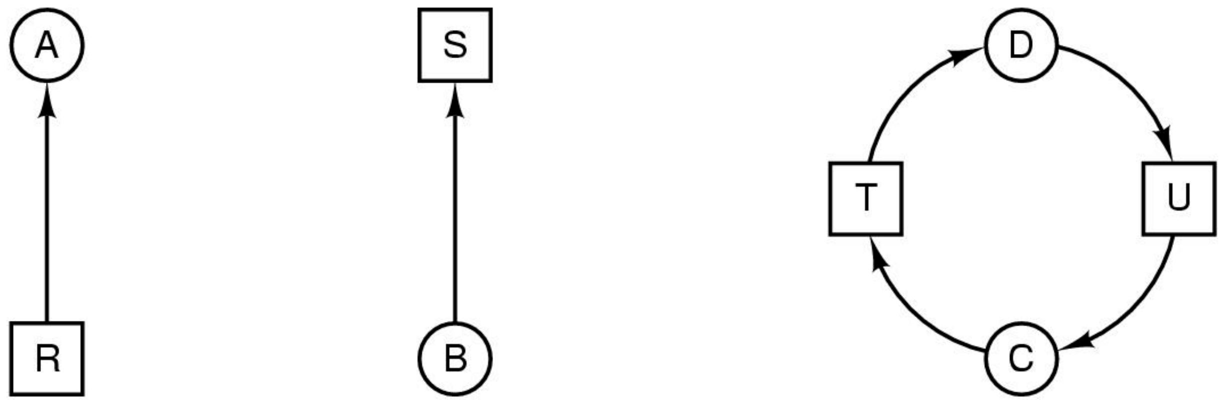
- **None** of the processes can …
  - **run**
  - **release** resources
  - be **awakened**

Four Conditions for Deadlock

1. Mutual exclusion condition **(互斥)**
   - each resource assigned to one process or is available
2. Hold and wait condition **(占有和等待)**
   - process holding resources can request additional
3. No preemption condition **(不可抢占)**
   - previously granted resources cannot forcibly taken away
4. Circular wait condition **(环路等待)**
   - must be a circular chain of two or more processes
   - each is waiting for resource held by next member of the chain

Deadlock Modeling

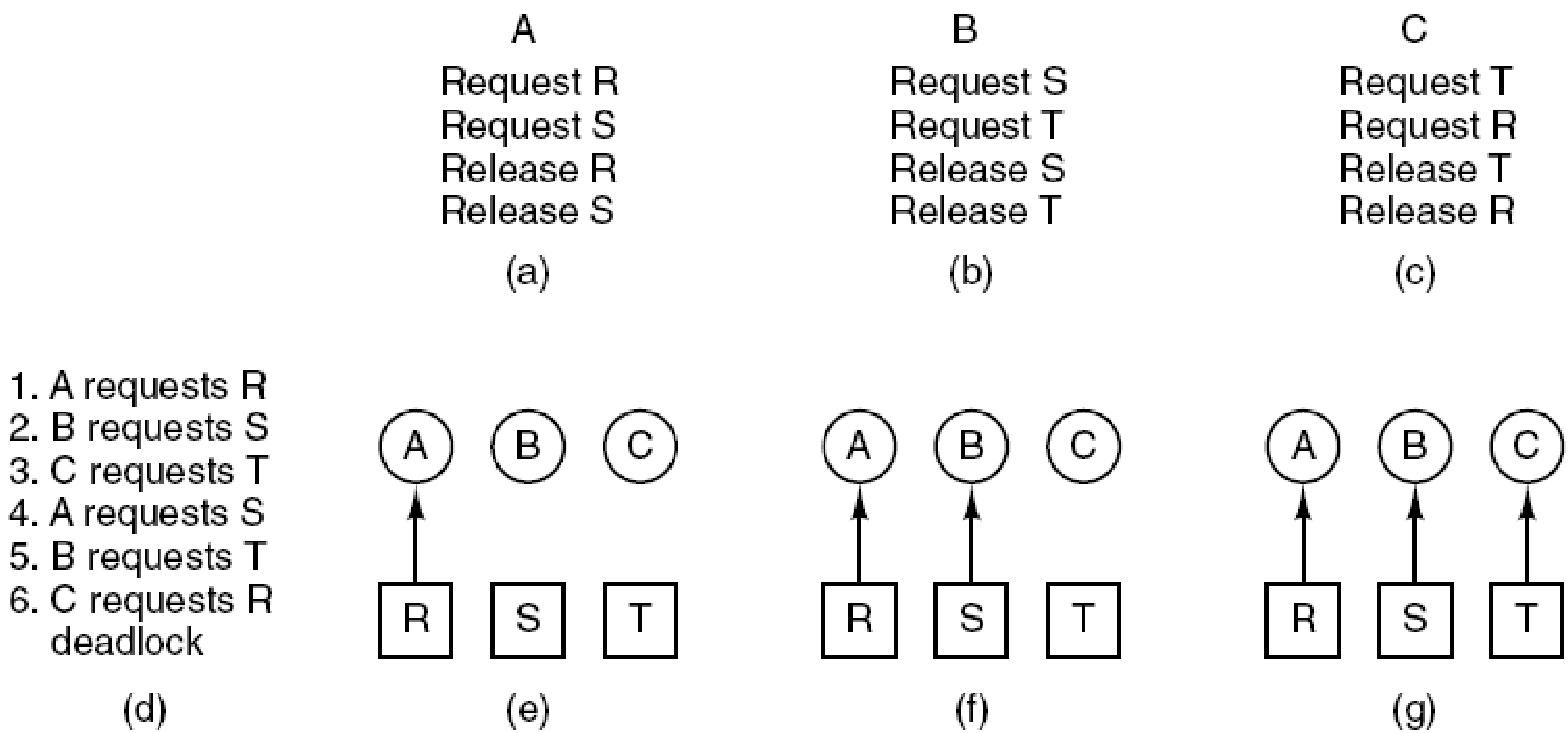
- Modeled with directed graphs (资源分配图)



(a) (b) (c)

  - resource R assigned to process A
  - process B is requesting/waiting for resource S
  - process C and D are in deadlock over resources T and U

Deadlock Modeling

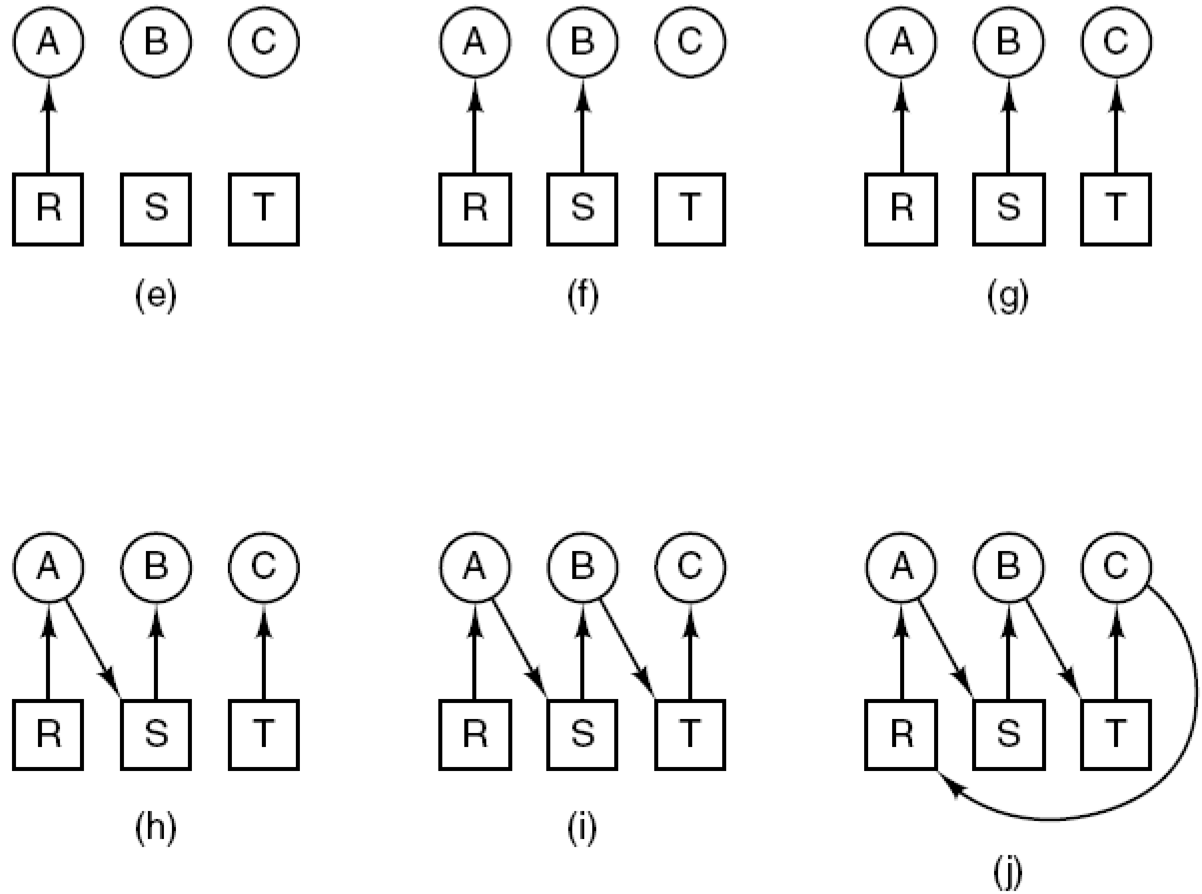


An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling

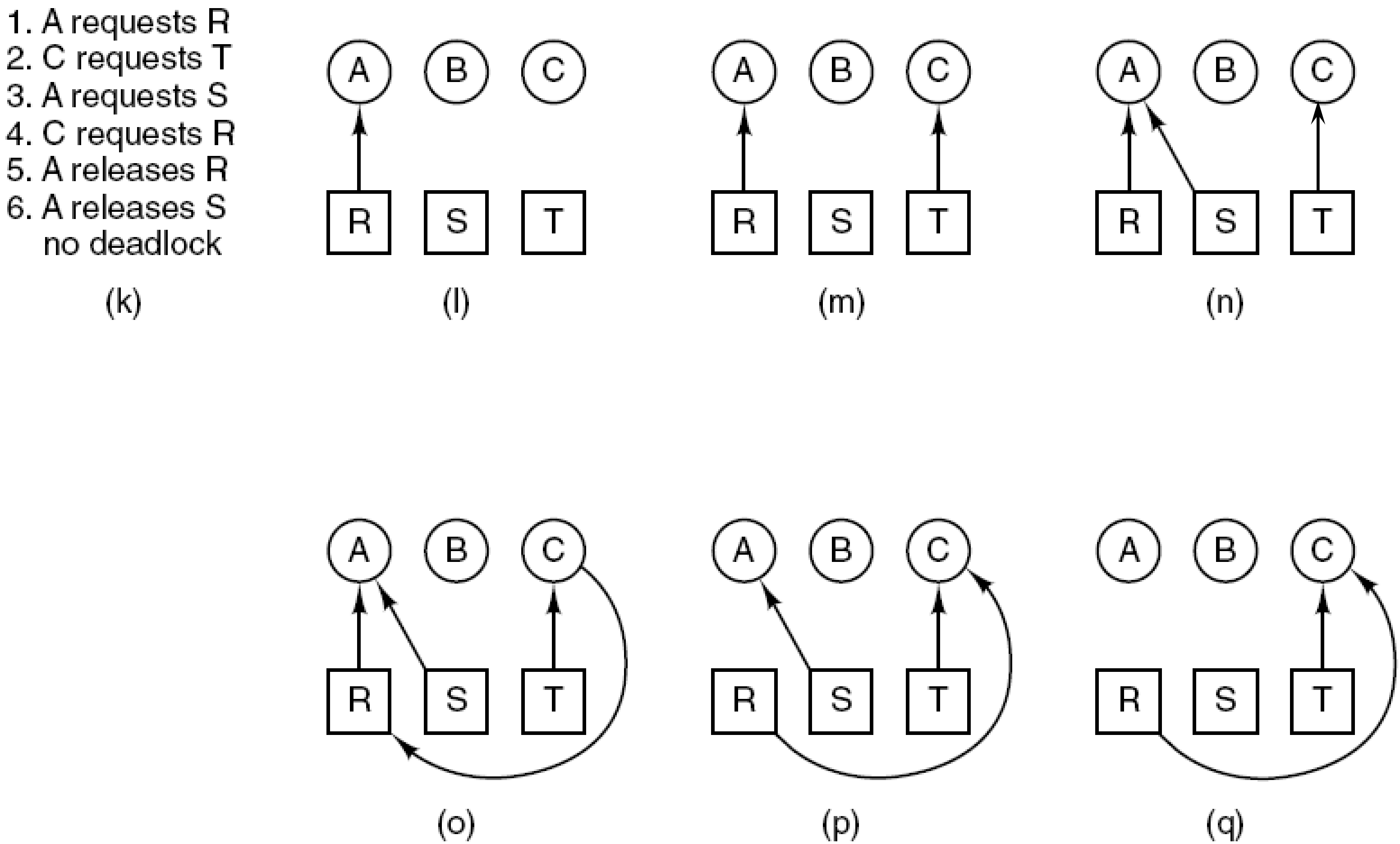
1. A requests R
2. B requests S
3. C requests T
4. A requests S
5. B requests T
6. C requests R
   deadlock

(d)



An example of how deadlock occurs
and how it can be avoided.

Deadlock Modeling



An example of how deadlock occurs and how it can be avoided.

Deadlock Modeling

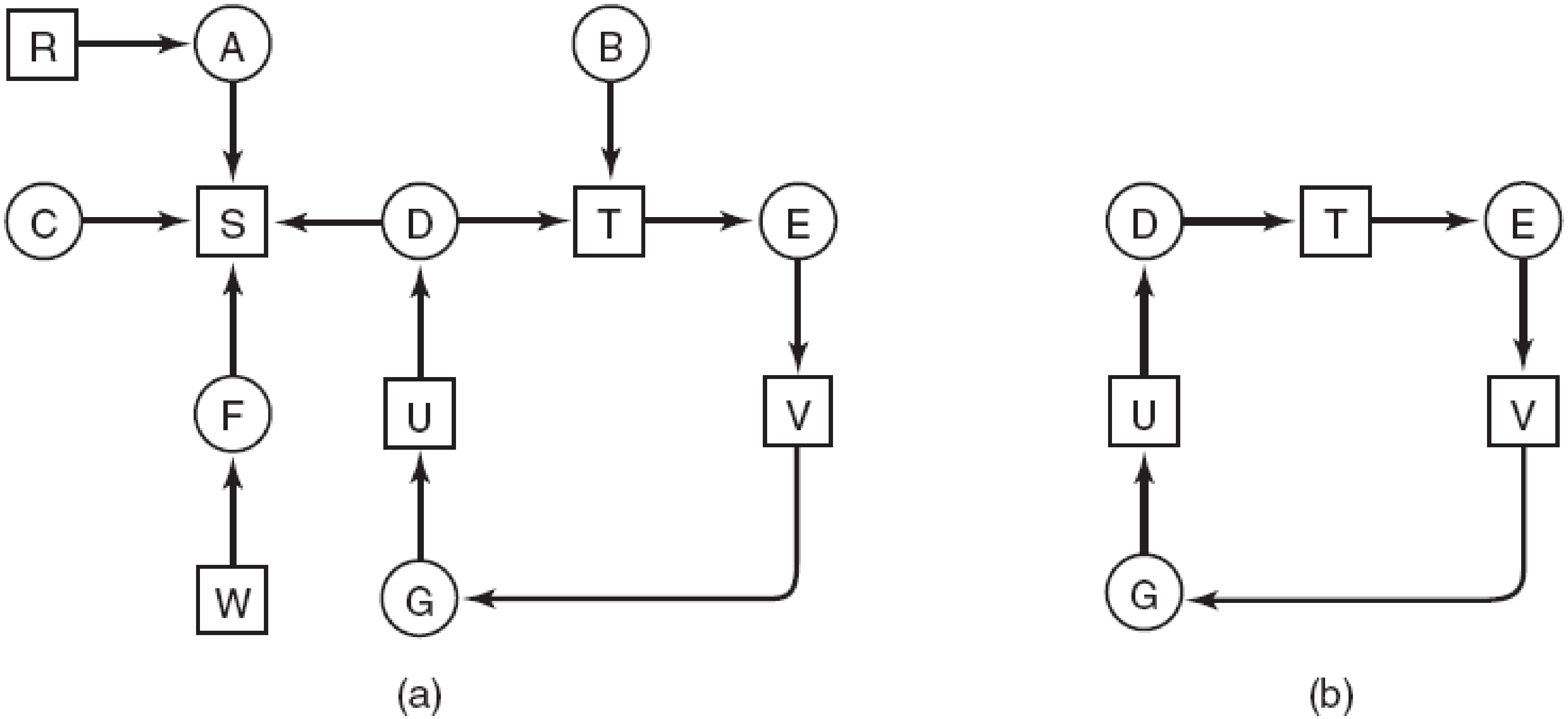
Strategies for dealing with deadlocks:

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

The Ostrich Algorithm

- **Pretend** there is no problem
- **Reasonable** if
  - deadlocks occur very rarely
  - cost of prevention is high
- **UNIX** and **Windows** takes this approach
- It is a trade off between
  - convenience
  - correctness

Deadlock Detection with One Resource of Each Type



(a) A resource graph. (b) A cycle extracted from (a).

Algorithm for detecting deadlock

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

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Current allocation matrix

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

Row n is current allocation
to process n

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

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 2 is what process 2 needs

$$\sum_{i=1}^{n} C_{ij} + A_j = E_j$$

The four data structures needed
by the deadlock detection algorithm.

Deadlock detection algorithm

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.

Deadlock Detection with Multiple Resources of Each Type

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

Current allocation matrix

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

Request matrix

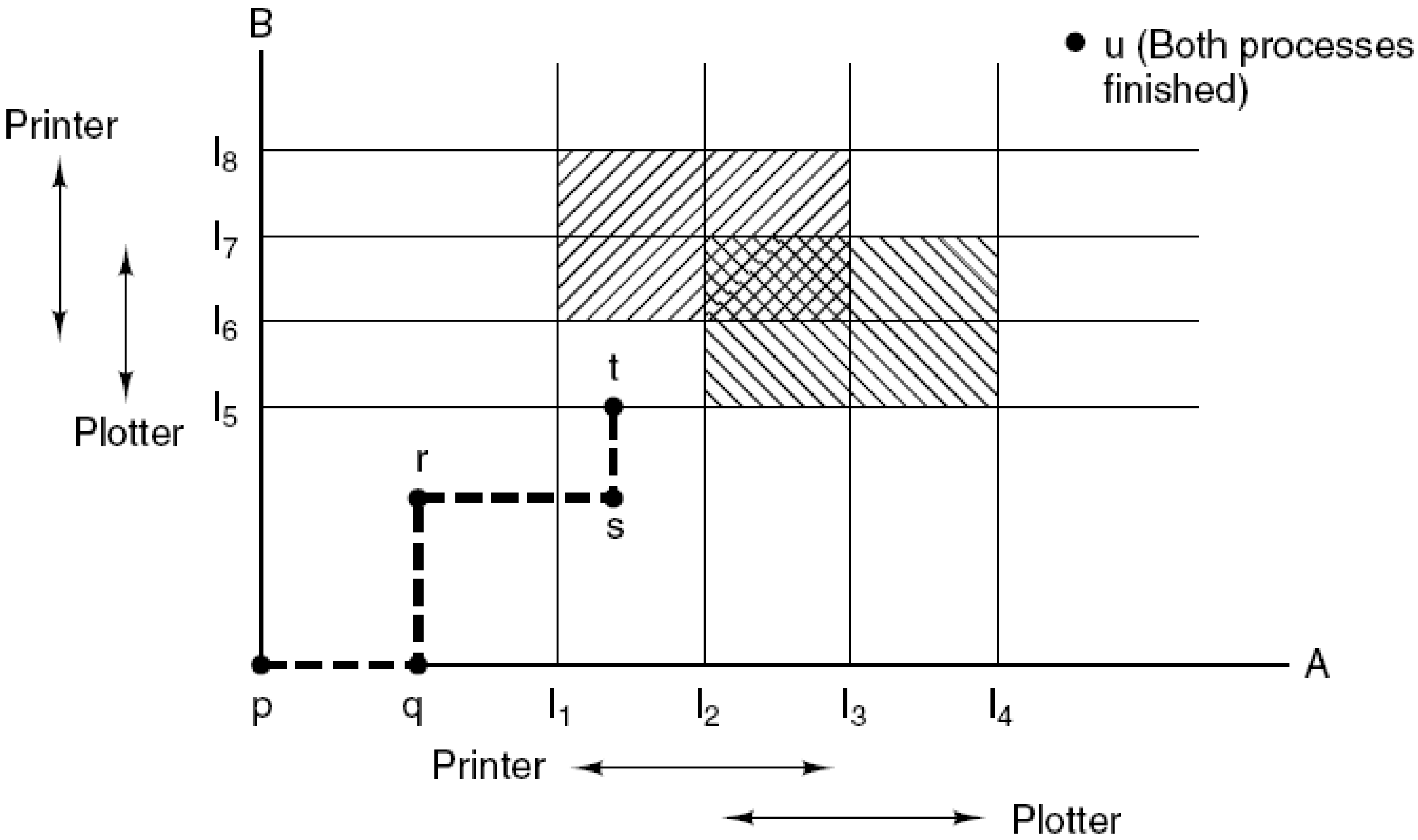
$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

An example for the deadlock detection algorithm.

Recovery from Deadlock

- Recovery through preemption (利用抢占)
  - take a resource from some other process
  - depends on nature of the resource
- Recovery through rollback (利用回退)
  - checkpoint a process periodically
  - use this saved state
  - restart the process if it is found deadlocked
- Recovery through killing processes (通过杀死进程)
  - crudest but simplest way to break a deadlock
  - kill one of the processes in the deadlock cycle
  - the other processes get its resources
  - choose process that can be rerun from the beginning

Deadlock Avoidance



Two process **resource trajectories**.(资源轨迹图)

Safe and Unsafe States

	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)

Demonstration that the state in (a) is safe.

Safe and Unsafe States

	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)

Demonstration that the state in (b) is not safe.

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)

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

Free: 1

(c)

Three resource allocation states:
(a) Safe. (b) Safe. (c) Unsafe.

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)

The banker's algorithm with multiple resources.

The Banker's Algorithm for Multiple Resources

Algorithm for checking to see if a state is safe:

1. Look for row, R, whose unmet resource needs 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).

Deadlock Prevention (预防)

- Attacking the mutual exclusion condition
  破坏互斥
- Attacking the hold and wait condition
  破坏占有和等待
- Attacking the no preemption condition
  破坏不可抢占
- Attacking the circular wait condition
  破坏循环等待

Attacking the Mutual Exclusion Condition

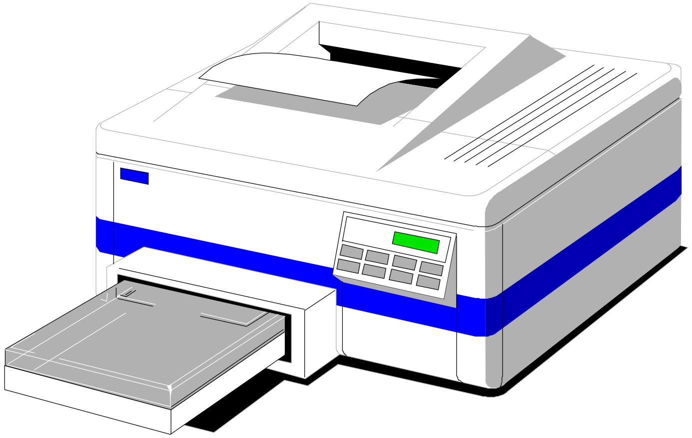
- Some devices (such as printer) can be spooled
  - only the printer daemon uses printer resource
  - thus deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
  - avoid assigning resource when not absolutely necessary
  - as few processes as possible actually claim the resource

Attacking the Hold and Wait Condition

- Require processes to request resources before starting
  - a process never has to wait for what it needs
- Problems
  - may not know required resources at start of run
  - also ties up resources other processes could be using
- Variation:
  - process must give up all resources
  - then request all immediately needed

Attacking the No Preemption Condition

- This is not a viable option
- Consider a process given the printer
  - halfway through its job
  - now forcibly take away printer
  - !!??



Attacking the Circular Wait Condition

(a) Numerically ordered resources.
(b) A resource graph.

Approaches to Deadlock Prevention

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

Summary of approaches to deadlock prevention.

Other Issues

- Two-phase locking
- Nonresource Deadlocks
- Livelock
- Starvation

Two-Phase Locking

- Phase One
  - process tries to lock all records it needs, one at a time
  - if needed record found locked, start over
  - (no real work done in phase one)

- If phase one succeeds, it starts second phase,
  - performing updates
  - releasing locks

Nonresource Deadlocks

- Possible for two processes to deadlock
  - each is waiting for the other to do some task

- Can happen with semaphores
  - each process required to do a *down()* on two semaphores (*mutex* and another)
  - if done in wrong order, deadlock results

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);
}
```

Busy waiting that can lead to livelock.

Starvation

- Algorithm to allocate a resource
  - may be to give to shortest job first
- Works great for multiple short jobs in a system
- May cause long job to be postponed indefinitely
  - even though not blocked
- Solution:
  - First-come, first-serve policy