CS 444 Operating Systems

Chapter 6 Deadlocks

J. Holly DeBlois

November 5, 2024

Resources to be Shared

- Hardware devices
- Software resources
  - A piece of information
  - Database records
- Preemptable
  - RAM
- Nonpreemptable
  - Printer, tape drive

Use a Semaphore to Protect Resources

- One resource

```
typedef int semaphore;
semaphore resource_1;

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

(a)

- Two resources

```
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)

A Potential Deadlock

- Deadlock-free

```
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)

- 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)

Deadlock Definition

- A set of processes is deadlocked if

1. Each process in the set is waiting for an event
2. That event can be caused only by another process in the set

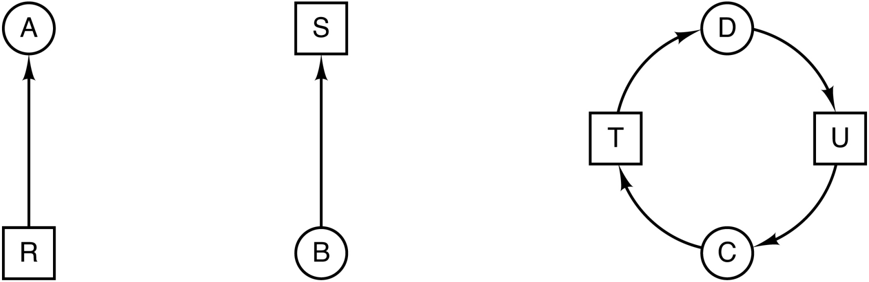
Conditions for Resource Deadlock

- Four conditions must hold

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

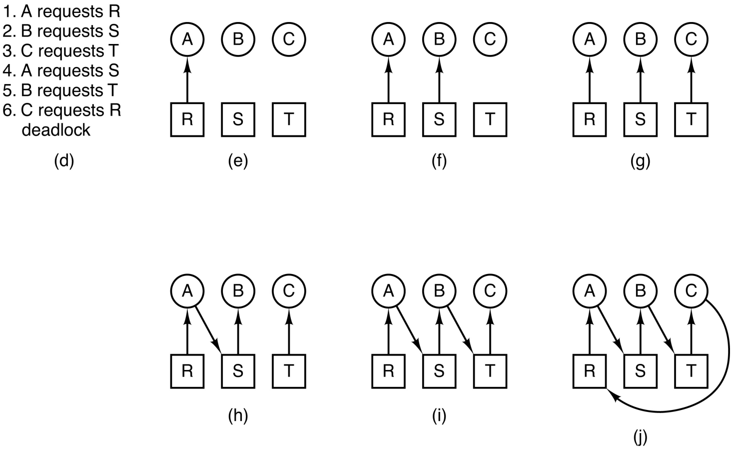
Resource Allocation Graph

- Holding a resource
- Requesting a resource
- Deadlock



(a) (b) (c)

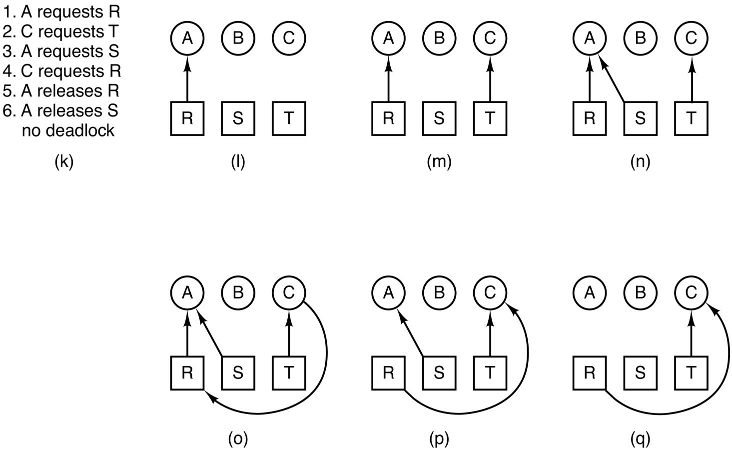
An Example of Circular Wait



- A, B, and C are in circular wait

Holding Process B Back

- Hold process B back to break up the cycle

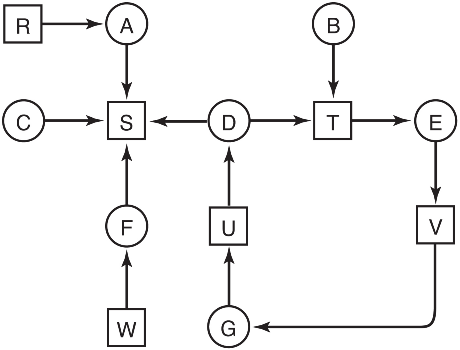


Strategies for Dealing with Deadlocks

- Ignore the problem — maybe it will go away
  - The ostrich algorithm
  - The current strategy used in most systems
- Detection and recovery
  - Let deadlocks occur, detect them, and take action
- Dynamic avoidance
  - Careful resource allocation
- Prevention
  - Structurally negating one of the four required conditions

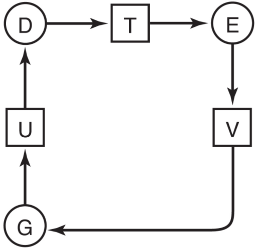
Deadlock Detection

- A resource graph



(a)

- A cycle extracted from the graph



(b)

DFS to Detect Deadlocks

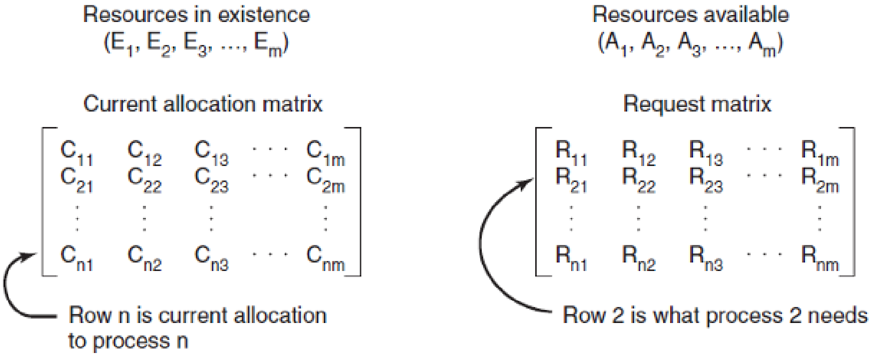
- For each node N in the graph, perform these steps with N as the current node

1. Initialize S to an empty stack and designate all edges as unmarked
2. Push the current node into S, check if the node appears in S twice
   - If yes, the graph has a cycle (listed in S) and thus a deadlock
3. If the current node has any unmarked outgoing edges, go to step 4; if not, go to step 5
4. Pick an unmarked outgoing edge, mark it and follow it to the new current node; go to step 2
5. If this is initial node, the graph does not contain cycles and no deadlocks. Otherwise, pop the node from S and go back to the previous node

When There Are Multiple Resources of Each Type

- The previous deadlock detection algorithm works with the assumption that there is just one resource of each type
- Often a computer has multiple resources of each type
- Use four data structures to support deadlock detection when multiple resources are available

Four Data Structures



Deadlock Detection Algorithm

1. Look for an unmarked process P_i for which the i-th row of R (request) is less than or equal to A (available)
   - This process can acquire all resources it needs for successful completion
2. If such a process is found, add the i-th row of C (current allocation) to A, mark the process, go back to step 1
   - Pretend this process has finished and releases its acquired resources
3. If no such process exists, algorithm terminates
   - The unmarked processes are in a deadlock

Example

	Tape drives	Plotters	Scanners	Blu-rays
E =	4	2	3	1

$$E = (\,4 \quad 2 \quad 3 \quad 1\,)$$

	Tape drives	Plotters	Scanners	Blu-rays
A =	2	1	0	0

$$A = (\,2 \quad 1 \quad 0 \quad 0\,)$$

Current allocation matrix

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

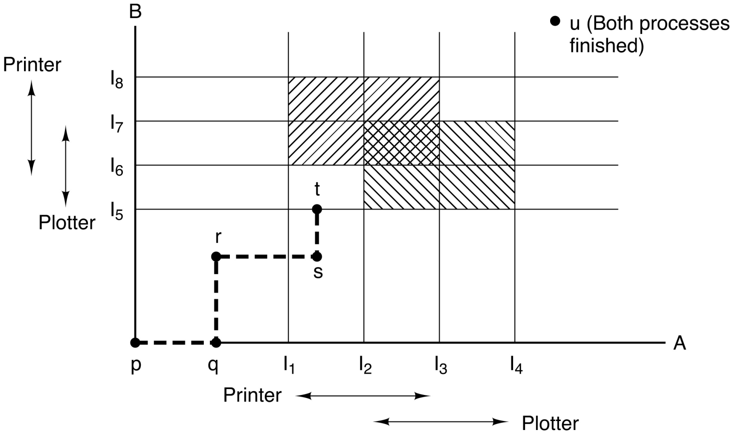
Request matrix

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

Recovery from Deadlock

- Possible methods of recovery, although none are “attractive”:
- Preemption
- Rollback
    - Checkpoints
- Killing processes

Resource Trajectories



- Two processes make requests for printer and plotter
- Avoid deadlock by following viable trajectories

Safe and Unsafe States

(a) Free: 3

	Has	Max
A	3	9
B	2	4
C	2	7

(b) Free: 1

	Has	Max
A	3	9
B	4	4
C	2	7

(c) Free: 5

	Has	Max
A	3	9
B	0	–
C	2	7

(d) Free: 0

	Has	Max
A	3	9
B	0	–
C	7	7

(e) Free: 7

	Has	Max
A	3	9
B	0	–
C	0	–

- The state in (a) is safe because
- Process B can get all it needs, finish, and release resources
- Then process C can finish
- Then process A can finish

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	Đ	Đ
C	2	7

Free: 4
(d)

- The state in (a) is safe
- The state in (b) is not safe

Banker's Algorithm for Single Resource

(a)

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

Free: 10

(b)

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

Free: 2

(c)

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

Free: 1

- The state in (a) is safe
- The state in (b) is safe
- The state in (c) is not safe

Banker's Algorithm for Multiple Resources

Process	Tape drives	Plotters	Printers	Blu-rays
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	Blu-rays
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 assigned

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

- 2 tables: current allocation, future need
- 3 vectors: total in existence E, present allocation P, available A

Banker's Algorithm

1. Look for a process S whose unmet resource needs are all smaller than or equal to A
   - If no such process exists, the system will eventually deadlock
2. Assume S requests all resources needed and finishes, mark S as terminated, return its resources to the vector A
3. Repeat steps 1 and 2 until
   1. Either all processes are marked terminated (safe state)
   2. Or no process is left whose resource needs can be met (deadlock)

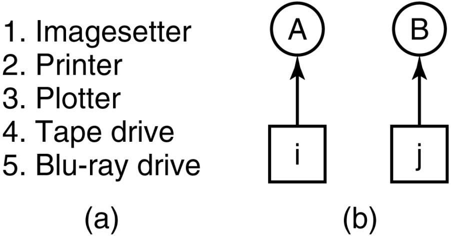
Deadlock Prevention

- Assure that at least one of conditions is never satisfied

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

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

Prevent Circular Wait



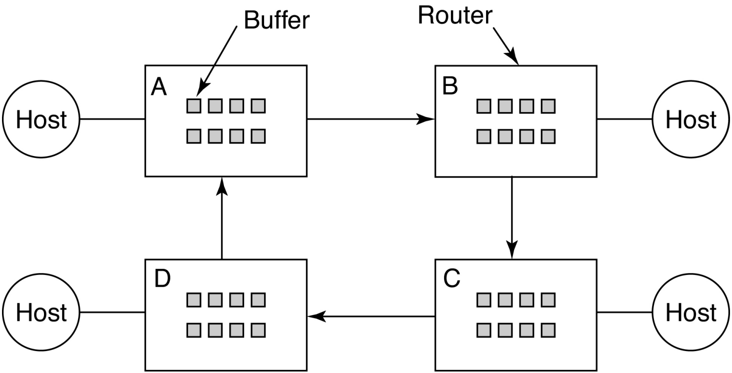
- Numerically ordered resources
- Request resources monotonically

Communication Deadlock

- Cooperation synchronization
  - Send/receive acknowledgment
  - Lost acknowledgment
- Competition synchronization

Communication Deadlock

- A deadlock in a network



Livelock

```
void process_A(void) {
    acquire_lock(&resource_1);
    while (try_lock(&resource_2) == FAIL) {
        release_lock(&resource_1);
        wait_fixed_time();
        acquire_lock(&resource_1);
    }
    use_both_resources( );
    release_lock(&resource_2);
    release_lock(&resource_1);
}

void process_A(void) {
    acquire_lock(&resource_2);
    while (try_lock(&resource_1) == FAIL) {
        release_lock(&resource_2);
        wait_fixed_time();
        acquire_lock(&resource_2);
    }
    use_both_resources( );
    release_lock(&resource_1);
    release_lock(&resource_2);
}
```

- Processes are not strictly blocked, but they are not going anywhere
- Compete for process table entries
- Compete for file table entries