Operating System Principles

操作系统原理

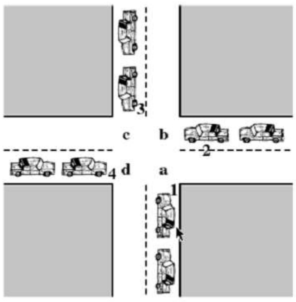
IPC: Deadlock

李旭东

leexudong@nankai.edu.cn

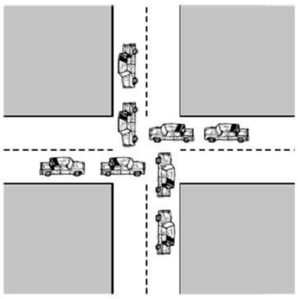
Nankai University

Case: Traffic



(a) Deadlock possible

Case: Traffic



(b) Deadlock

Objectives

- Deadlock Definition
- Deadlock Conditions
- Deadlock Modeling
- Deadlock Detection
- Deadlock Recovery
- Deadlock Avoidance
- Deadlock Prevention

Deadlock

- Deadlock
  - A situation in which two or more competing actions are each waiting for the other to finish, and thus neither ever does
  - 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.
  - Deadly-Embrace

The Reason of Deadlock

- Deadlock
  - Competing Resource
  - Running Order of a Set of Processes

Resource

- Resource which causes deadlock
- hardware, software, information
- Preemptable, Nonpreemptable

Nonpreemptable Resources

- Sequence of events required to use a resource:
  - Request the resource
  - Use the resource
  - Release the resource

Resource Acquisition

```
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)

Which situation will be a 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);
}
```

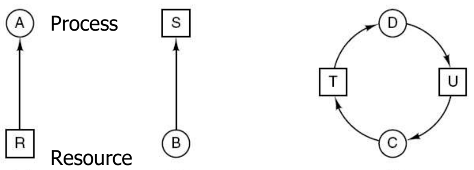
(a)

```
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_1);
    down(&resource_2);
    use_both_resources();
    up(&resource_2);
    up(&resource_1);
}
```

(b)

Deadlock Model

SRAG(System Resource Allocation Graph)

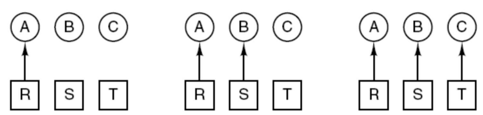


(a) (b) (c)

SRAG=(V,E)

Deadlock Model

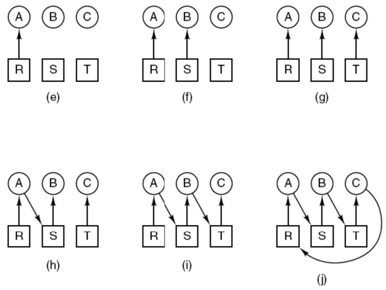
A	B	C
Request R Request S Release R Release S	Request S Request T Release S Release T	Request T Request R Release T Release R
(a)	(b)	(c)



(e) (f) (g)

Deadlock Model

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

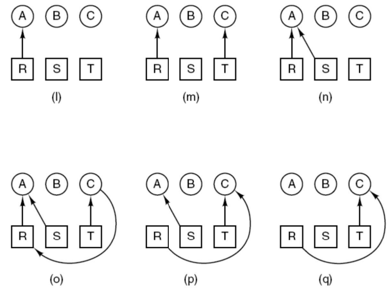


Deadlock!

Deadlock Model

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

(k)



No Deadlock!

Necessary Conditions

- Coffman et al.,1971
  - Mutual exclusion
  - Hold and wait
  - No preemption
  - Circular wait

deadlock depend on a policy

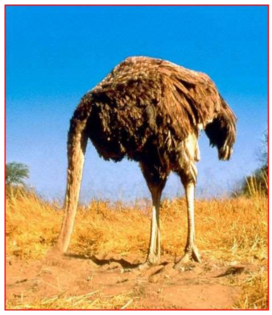
- ?one resource can be shared among multi-procceses
- ?a process which holds a resource requests another resource
- ?policy which allows circular wait
- ?multi-processes cocurrent

Strategies for Deadlocks

- Strategies for dealing with deadlocks:
  - Detection and recovery
    - let deadlocks occur, detect them, take action.
  - Dynamic avoidance
    - by careful resource allocation
  - Prevention
    - by structurally negating one of the four required conditions
  - ...

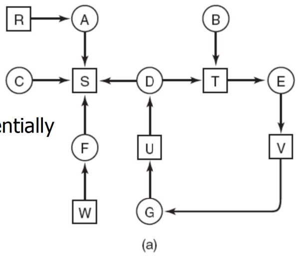
Strategies for Deadlocks

- the ostrich algorithm Just ignore the problem

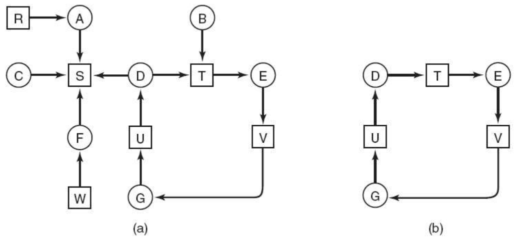


Deadlock Detection with One Resource of Each Type (1)

?Deadlock Potentially



Deadlock Detection with One Resource of Each Type (2)



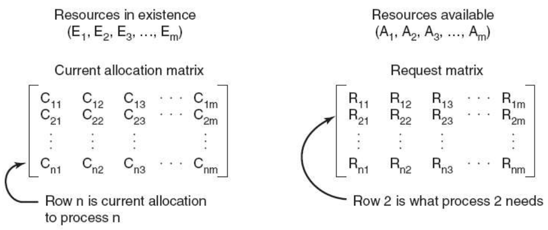
Deadlock Detection with One Resource of Each Type

- 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.
  - …

Deadlock Detection with One Resource of Each Type

- 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 (1)



$$\left[\begin{array}{ccccc} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{array}\right] \qquad \left[\begin{array}{ccccc} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{array}\right]$$

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

Deadlock Detection with Multiple Resources of Each Type (2)

- Deadlock detection algorithm:
  - 1. Look for an unmarked process, Pi , 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 (3)

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

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

Current allocation matrix

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

Request matrix

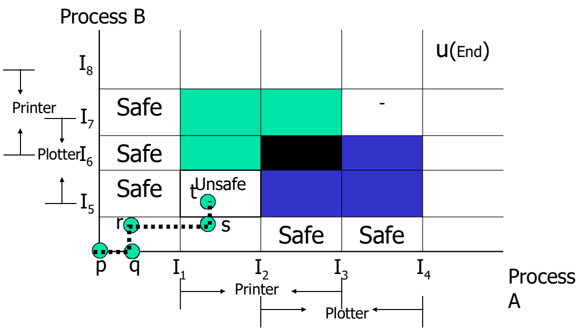
$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

Recovery from Deadlock

- Recovery through preemption
- Recovery through rollback
- Recovery through killing processes

Deadlock Avoidance

Resource Trajectories (资源轨迹图)



Safe and Unsafe States (1)

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3
(a)

If ...

	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)

Safe and Unsafe States (2)

	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)

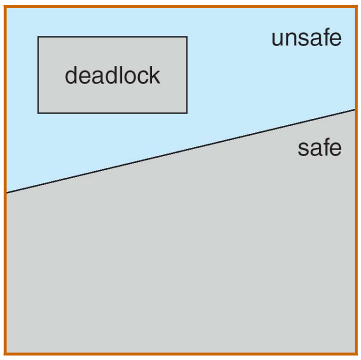
	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0
(d)

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7
(e)

Safe and Unsafe States (3)



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)

The Banker's Algorithm for Multiple Resources (1)

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 for Multiple Resources (2)

- 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).

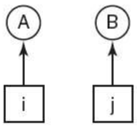
Deadlock Prevention

- Attacking the mutual exclusion condition
- Attacking the hold and wait condition
- Attacking the no preemption condition
- Attacking the circular wait condition

Attacking the Circular Wait Condition

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

(a)



(b)

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

Order resources numerically

- Ri (1,…,n)
- f(Ri): the id of Ri
- How to prove
  - If deadlock, so circular wait
  - i.e. existed:
    - ***f(R1)***< f(R2)<…< f(Rn)<***f(R1)***
  - ***contradiction***

Other Issues

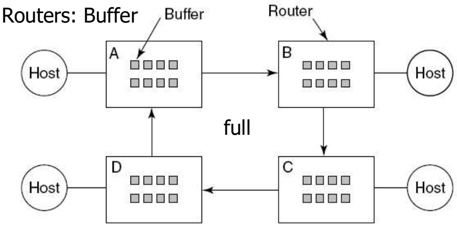
- Two-phase locking
- Communication deadlocks
- Livelock
- Starvation

Two-phase locking

- In the first phase, the process tries to lock all the records it needs, one at a time.
  - If it succeeds, it begins the second phase, performing its updates and releasing the locks. No real work is done in the first phase.
- If during the first phase, some record is needed that is already locked
  - the process just releases all its locks and starts the first phase all over.

Communication deadlocks

Hosts
Routers: Buffer



communication systems

Livelock

Case: Each one needs two resources and they use the polling primitive enter_region to try to acquire the necessary locks. If the attempt fails, the process just tries again.

```
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

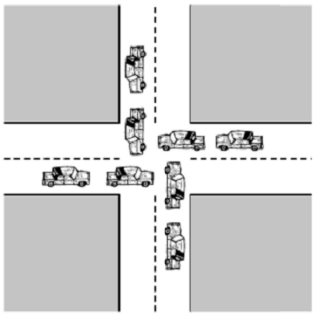
Starvation

- Starvation
  - In a dynamic system, requests for resources happen all the time.
  - Some policy is need ed to make a decision about who gets which resource when.
  - This policy, although seemingly reasonable, may lead to some processes never getting service even though they are not deadlocked.
- Case:
  - SJF

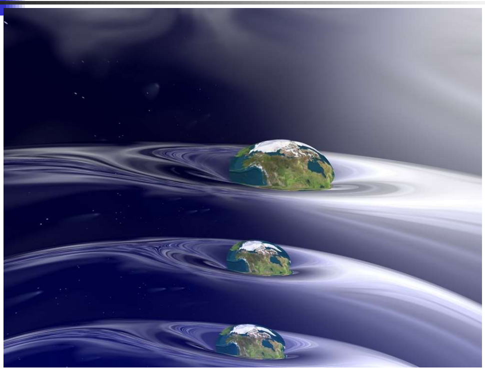
Deadlock, livelock, starvation

critical section	A section of code within a process that requires access to shared resources and which may not be executed while another process is in a corresponding section of code.
deadlock	A situation in which two or more processes are unable to proceed because each is waiting for one of the others to do something.
livelock	A situation in which two or more processes continuously change their state in response to changes in the other process(es) without doing any useful work.
mutual exclusion	The requirement that when one process is in a critical section that accesses shared resources, no other process may be in a critical section that accesses any of those shared resources.
race condition	A situation in which multiple threads or processes read and write a shared data item and the final result depends on the relative timing of their execution.
starvation	A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

?Deadlock



Parallel Universes



Summary

- Deadlock Definition
- Deadlock Conditions
- Deadlock Modeling
- Deadlock Detection
- Deadlock Recovery
- Deadlock Avoidance
- Deadlock Prevention

Any Questions?

Homework

- Give some examples to explain the differences between deadlock, livelock and starvation