CS323 Operating Systems
Deadlock

Yuanyuan Zhou

Lecture 11

2/14/2003

Content of this lecture

- Administrative announcements
- Resource
- Deadlock
- Deadlock prevention
- Summary

Administrative

- None

3

2/13/2003

CS 323 - Operating Systems, Yuanyuan Zhou

Review: IPC

- Data races
- Critical regions and mutual exclusions
- Solutions:
  - Peterson's solution
  - TSL
  - Semaphores & Mutex
  - Monitor
  - Barrier
  - Message Passing
- Classic IPC Problems

4

2/13/2003

CS 323 - Operating Systems, Yuanyuan Zhou

Resource (1)

- A **resource** is a commodity needed by a process.
- Resources can be either:
  - **serially reusable:** e.g., CPU, memory, disk space, I/O devices, files.
    acquire → use → release
  - **consumable:** produced by a process, needed by a process; e.g., messages, buffers of information, interrupts.
    create → acquire → use
    Resource ceases to exist after it has been used, so it is not released.

Resource (2)

- Resources can also be either:
  - **preemptible**: e.g., CPU, central memory or
  - **non-preemptible**: e.g., tape drives.
- And resources can be either:
  - **shared** among several processes or
  - **dedicated** exclusively to a single process.

Using Semaphore to Share Resource

```
   Process P();
2  { A.Down();
3    B.Down();
       use both resource
4    B.Up();
5    A.Up(); }
```

```
0  Process Q();
6  { A.Down();
     B.Down();
       use both resource
     B.Up();
     A.Up(); }
```

External Semaphore A(1), B(1);

2 External Semaphore A(0), B(1);

3 External Semaphore A(0), B(0);

4 External Semaphore A(0), B(1);

5 External Semaphore A(1), B(1);

But Deadlock can Happen!

```
1  Process P();
2  { A.Down();
     B.Down();
       use b
     B.Up()
     A.Up()
```

```
1  Process Q();
3  { B.Down();
     A.Down();
       use both resources
     A.Up();
     B.Up(); }
```

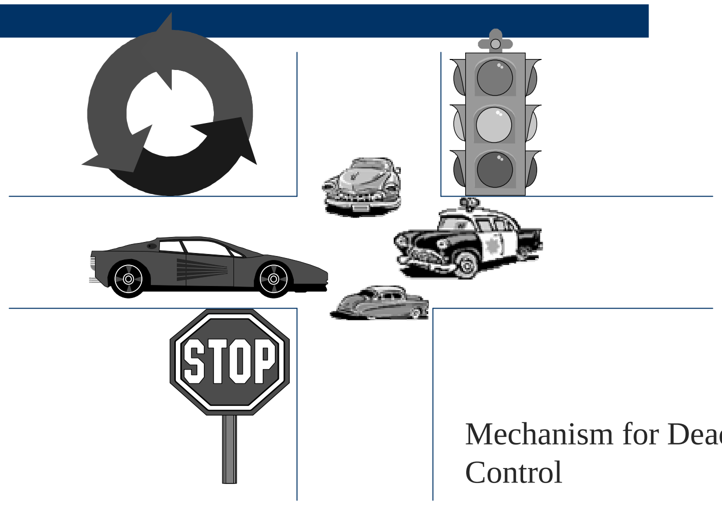
DEADLOCK

1 External Semaphore A(1), B(1);

2 External Semaphore A(0), B(1);

3 External Semaphore A(0), B(0);

Deadlock



Mechanism for Deadlock Control

Deadlock Definition

- What is a deadlock?
  - A process is **deadlocked** if it is waiting for an event that will never occur.
  - Typically, but not necessarily, more than one process will be involved together in a deadlock (the *deadly embrace*).

- Is deadlock the same as starvation (or indefinitely postponed)?
  - A process is **indefinitely postponed** if it is delayed repeatedly over a *long* period of time while the attention of the system is given to other processes. I.e., logically the process may proceed but the system never gives it the CPU.

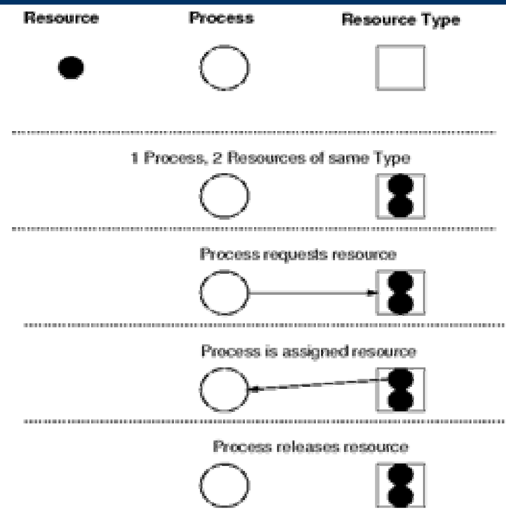
Conditions for Deadlock

- What conditions should exist in order to lead to a deadlock
- Group discussion (2 minutes)
  - Can use real life analogy such as
    - “You take the monitor, I grab the keyboard”
    - Cars in intersection

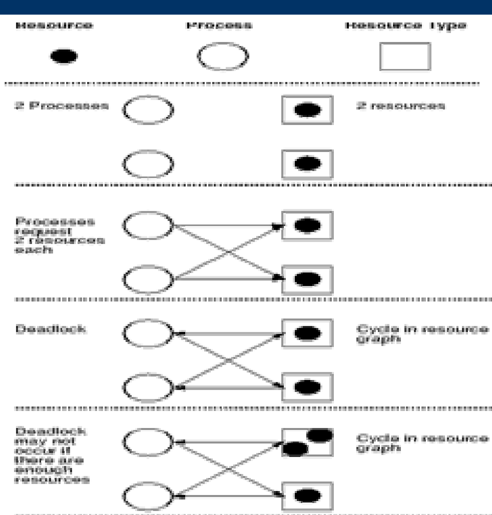
Necessary and Sufficient Conditions for Deadlock

- **Mutual exclusion**
  - Processes claim **exclusive** control of the resources they require
- **Wait-for condition**
  - Processes hold resources already allocated to them while waiting for additional resources
- **No preemption condition**
  - Resources cannot be removed from the processes holding them until used to completion
- **Circular wait condition**
  - A circular chain of processes exists in which each process holds one or more resources that are requested by the next process in the chain

Resource Allocation Graph



Deadlock Model



Deadlock Issues

- **Prevention**
  - design a system in such a way that deadlocks cannot occur, at least with respect to serially reusable resources.
- **Avoidance**
  - impose less stringent conditions than for prevention, allowing the possibility of deadlock, but sidestepping it as it approaches.
- **Detection**
  - in a system that allows the possibility of deadlock, determine if deadlock has occurred, and which processes and resources are involved.
- **Recovery**
  - after a deadlock has been detected, clear the problem, allowing the deadlocked processes to complete and the resources to be reused. Usually involves destroying the affected processes and starting them over.

The Ostrich Algorithm

- Don't do anything, simply restart the system (stick your head into the sand, pretend there is no problem at all).
- Rational: make the common path faster and more reliable
  - Deadlock prevention, avoidance or detection/recovery algorithms are expensive
  - if deadlock occurs only rarely, it is not worth the overhead to implement any of these algorithms.

17

Group Discussion

- How do we prevent deadlocks?
  - You can use real life analogies
  - If you cannot give an answer, you are like an “ostrich”?

18

Deadlock Prevention: Havender's Algorithms

- Break one of the deadlock conditions.
  - *Mutual exclusion*
    - Solution: exclusive use of resources is an important feature, but for some resources (virtual memory, virtual disks, CPU), it is possible.
  - *Hold-and-Wait condition*
    - Solution: Force each process to request all required resources at once. It cannot proceed until all resources have been acquired.
  - *No preemption condition*
    - Solution: If a process holding some reusable resources makes a further request which is denied, and it wishes to wait for the new resources to become available, it must release all resources currently held and, if necessary, request them again along with the new resources. Thus, resources are removed from a process holding them.
  - *Circular wait condition*
    - Solution: All resource types are numbered. Processes must request resources in numerical order; if a resource of type is held, the only resources which can be requested must be of types .

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
- Note similarity to requesting all resources at once
- Algorithm works where programmer can arrange program can be stopped, restarted

Break Circular Wait Condition

- Request one resource at a time. Release the current resource when request the next one
- Global ordering of resources
  - Requests have to made in increasing order
  - Req(resource1), req(resource2)..
  - Why no circular wait?

Summary: Deadlock Prevention

condition	How to break it
Mutual Exclusion	Spool everything
Hold and wait	Request all resources initially
No preemption	Take resources away
Circular wait	Order resources numerically

Reminder

- Next lecture: Synchronization (chapter 2.3 & 2.4)
- Quiz1 due today at 5pm(only 1 try)
- MP1