

# 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

Oct. 24, 2007

CSE660: Introduction to Operating Systems

1

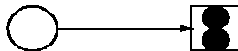
# Resource Allocation Graph



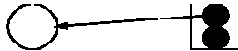
1 Process, 2 Resources of same Type



Process requests resource



Process is assigned resource



Process releases resource

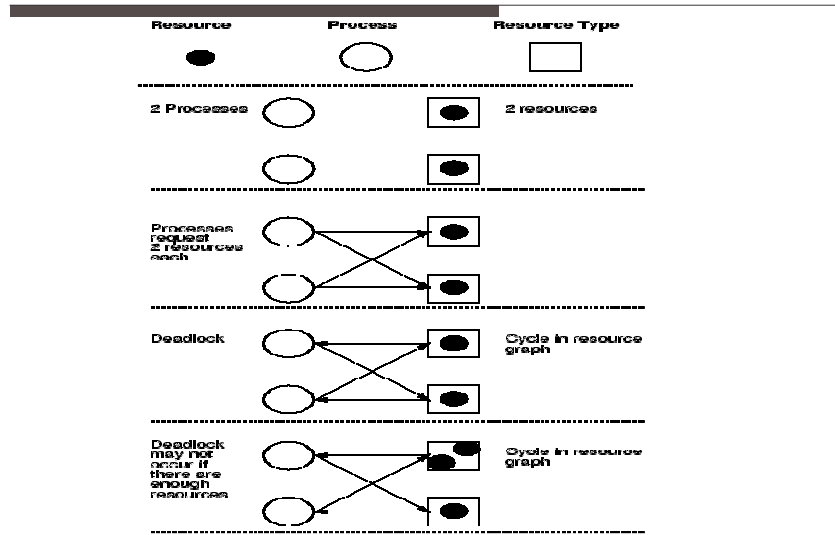


Oct. 24, 2007

CSE660: Introduction to Operating Systems

2

# Deadlock Model



Oct. 24, 2007

CSE660: Introduction to Operating Systems

3

# How To Deal With Deadlock?

## □ 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.

Oct. 24, 2007

CSE660: Introduction to Operating Systems

4

## The Ostrich Algorithm

---

- Guess: what is implemented in Linux?
  
- Don't do anything, simply restart the system (stick your head into the sand, pretend there is no problem at all).
- Rationale:
  - 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.

## 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, 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: forcibly take away the resources assigned to the process due to lack of other requested resources.
  - *Circular wait condition*
    - Solution: All resource types are numbered. Processes must request resources in numerical order

## 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
- Problems?

Oct. 24, 2007

CSE660: Introduction to Operating Systems

7

## Break Circular Wait Condition

---

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

Oct. 24, 2007

CSE660: Introduction to Operating Systems

8

## Summary: Deadlock Prevention

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

## Questions

Design an algorithm to detect deadlocks?

- Single resource per type
- Multiple resources per type

How to avoid deadlocks?

How to recover from deadlocks?

## Deadlock Detection

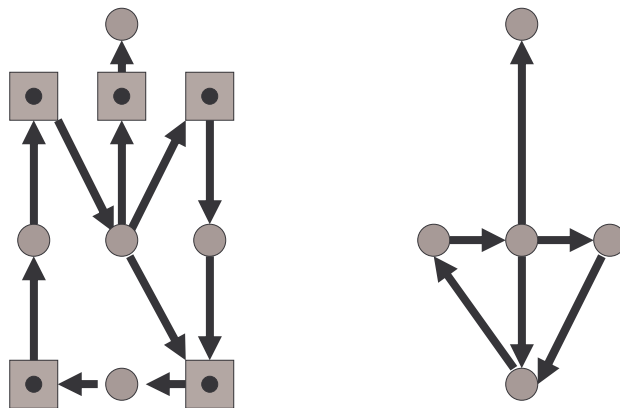
- Check to see if a deadlock has occurred!
- Single resource per type
  - Can use wait-for graph
  - Check for cycles
    - How?
- How about multiple resources per type?

Oct. 24, 2007

CSE660: Introduction to Operating Systems

11

## Wait for Graphs



Resource Allocation Graph    Corresponding Wait For Graph

Oct. 24, 2007

CSE660: Introduction to Operating Systems

12

## Deadlock Avoidance

---

- The system needs to know the resource ahead of time
- Banker Algorithm (Dijkstra, 1965)
  - Each customer tells banker the maximum number of resources it needs
  - Customer borrows resources from banker
  - Customer returns resources to banker
  - Customer eventually pays back loan
  - Banker only lends resources if the system will be in a *safe state* after the loan
- Safe state* - there is a lending sequence such that all customers can take out a loan
- Unsafe state* - there is a possibility of deadlock

Oct. 24, 2007

CSE660: Introduction to Operating Systems

13

## Recovery From Deadlock

---

- OPTIONS:
  - Kill deadlocked processes and release resources
  - Kill one deadlocked process at a time and release its resources
  - Rollback all or one of the processes to a checkpoint that occurred before they requested any resources
  
- Note: may have starvation problems

Oct. 24, 2007

CSE660: Introduction to Operating Systems

14

## Deadlock Summary

---

- In general, deadlock detection or avoidance is expensive
- Must evaluate cost of deadlock against detection or avoidance costs
- Deadlock avoidance and recovery may cause starvation problems
- Unix and Windows use Ostrich Algorithm