



Deadlock Handling

Lecture 22

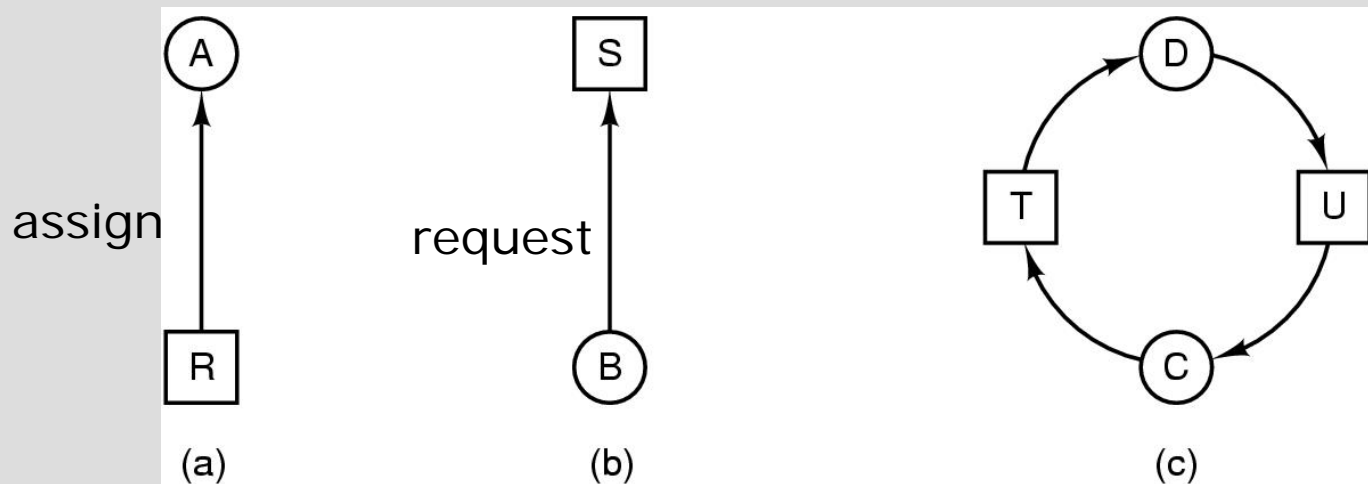
Klara Nahrstedt

CS241 Administrative

- Read Stallings Chapter 6 about Deadlock Handling
- Regular Quiz 7 this week on Queuing and Deadlock Handling
- Discussion Sessions this week (on Queuing and File Systems – needed for LMP1)
- LMP1 (Long Machine Problem) starts today
 - LMP1 is part of three major assignments (LMP1, LMP2, LMP3 which together will build distributed file system services)
 - LMP1 will be split into two parts, PART I and overall LMP1 delivery.
 - You will need to deliver PART I by Wednesday, March 28 and the final overall LMP1 will be due by April 2. For each delivery you will receive points.
 - LMP1 quiz will be on April 2.
 - The graders will provide some feedback on the Part I that should assist you in the overall LMP1 delivery.

Review: Deadlock Modeling

Modeled with directed graphs



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

A
Request R
Request S
Release R
Release S

(a)

B
Request S
Request T
Release S
Release T

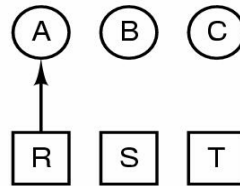
(b)

C
Request T
Request R
Release T
Release R

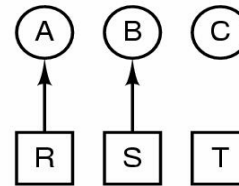
(c)

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

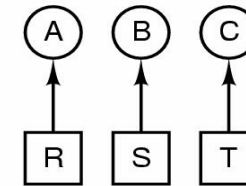
(d)



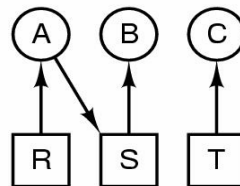
(e)



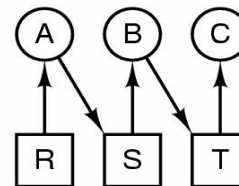
(f)



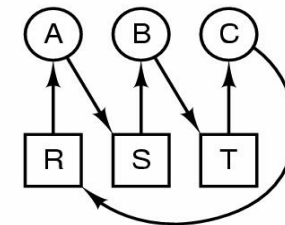
(g)



(h)



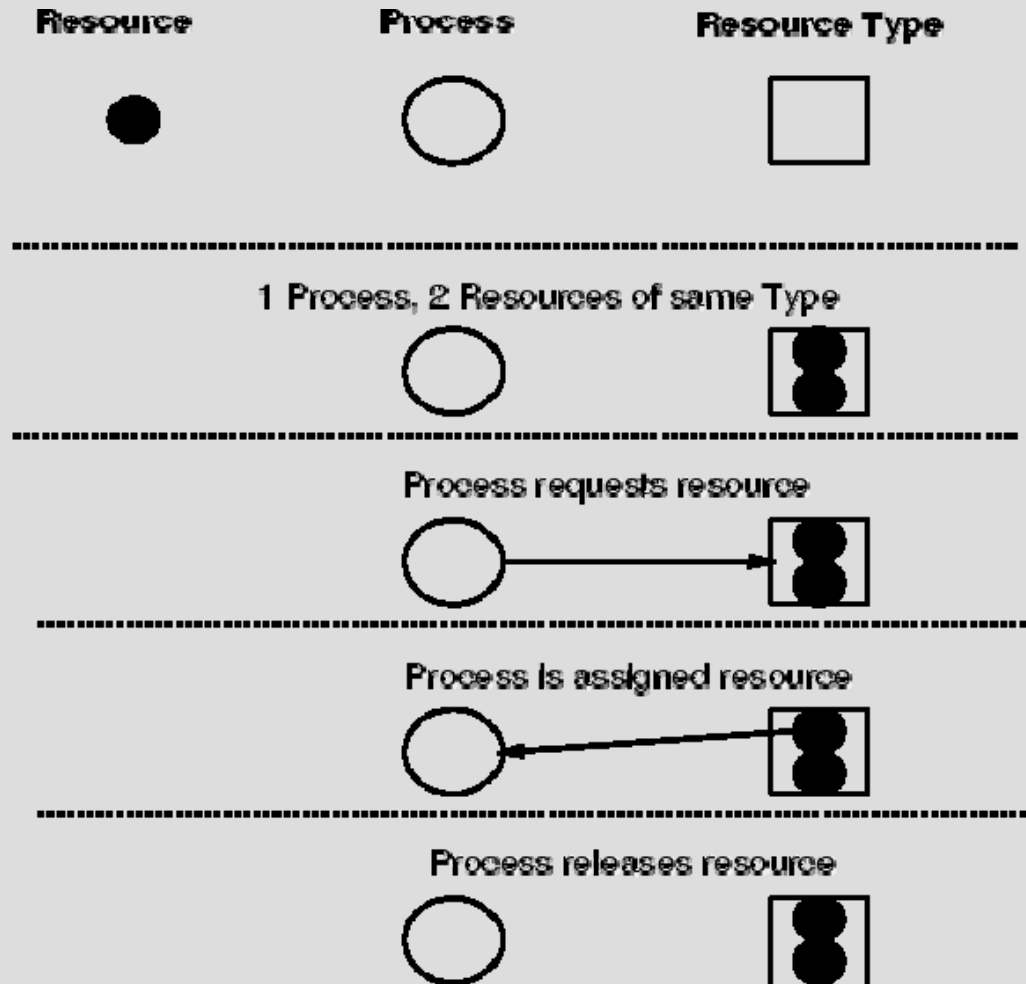
(i)



(j)

How deadlock occurs

Resource Allocation Graph (multiple resource types)



Strategies

Strategies for dealing with Deadlocks

prevention

negating one of the four necessary conditions

dynamic avoidance

careful resource allocation

detection and recovery

Deadlock Prevention

Prevention: design a system in such a way that deadlocks cannot occur, at least with respect to **serially reusable resources**

Having rules so that a deadlock can NEVER happen

Deadlock Prevention: Havender's Algorithms

If any one of the conditions for deadlock (with reusable resources) is denied, deadlock is impossible

Mutual Exclusion

We don't want to deny this; exclusive use of resources is an important feature. However, sometimes its possible---

- Virtual memory

- Virtual network connections

- Virtual disk

- Caches

- CPU preemption

Wait-for Condition

Force each process to request all required resources at once. It cannot proceed until all resources have been acquired

```
While did not get all resources
{
    P(mutex)
    Try to pick up all resources
    if did not get all resources put resources back
    V(mutex)
    wait a while and try again
}
Use resources
P(mutex)
Release all resources
V(mutex)
```

No Preemption Condition

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

```
while need a resource
{
  P(mutex)
  Test to see if resource is free
  if free, grab resource else release all resources
  V(mutex)
  Wait a while and try again
}
```

Circular Wait Condition

All resource types are numbered

Processes must request resources in numerical order

if a resource of order N is held, the only resources which can be requested must be of **order $> N$**

Process 1

P(s1)
P(s2)
Work
V(s2)
V(s1)

Process 2

P(s2)
P(s3)
Work
V(s3)
V(s2)

Process 3

P(s1)
P(s3)
Work
V(s3)
V(s1)

Deadlock Avoidance

Avoidance: impose less stringent conditions than for prevention, allowing the possibility of deadlock, but sidestepping it as it approaches

Banker algorithm

Each process provides the maximum number of resources for each type

For each request, the system checks if granting this request **may** lead to a deadlock in the worse case

If yes, not granting the request

If no, granting the request

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

Safe State and Unsafe State

Safe State

there is some scheduling order in which every process can run to completion even if all of them suddenly request their maximum number of resources immediately

From safe state, the system can guarantee that all processes will finish

Unsafe state: no such guarantee

Not deadlocked state

Some process may be able to complete

Example (Safe Sequence)

Processes	Res. Alloc		Max		Need		Available (Banker)	
	A	B	A	B	A	B	A	B
P1	3	0	5	3	2	3	6	5
P2	2	2	6	5	4	3		
P3	1	3	9	8	8	5		
P4	0	5	8	5	8	0		

Sequence:

- **P1 can go** (borrows from Banker A(2) and B(3), since $\text{Need.A}(2) < \text{Available.A}(6)$ and $\text{Need.B}(3) < \text{Available.B}(5)$. After lending, Available will be A(4), B(2)). After P1 finishes it releases Allocated Resources A(3) and B(0) to the banker, and so the banker will have $A(8) = A(6) + A(3)$ and $B(5) = B(5) + B(0)$ available
- **P2 can go** (borrows from Banker A(4) and B(3)). After P2 finishes it releases the allocated resource A(1) and B(3) to the banker and available will be A(11), B(7)
- **P3 can go** (borrows from Banker A(8) and B(5), since $\text{Need.A}(8) < \text{Available.A}(11)$ and $\text{Need.B}(5) < \text{Available.B}(7)$). After P3 finishes, it releases the allocated resources to the banker and Available resources will be $A(12) = A(11) + A(1)$ and $B(10) = B(7) + B(3)$
- **P4 can go** because P4 can borrow from Banker $\text{Need.A}(8) < \text{Available.A}(12)$ and $\text{Need.B}(0) < \text{Available.B}(10)$. After P4 finishes, the resources are released, and Banker will have available A(12) and B(15)
- Hence, there exists a safe sequence with this snapshot of resources and processes. The safe sequence is P1, P2, P3, P4

Example (Unsafe State) – let's assume Max/Need changes for the processes

Processes	Res. Alloc		Max		Need		Available (Banker)	
	A	B	A	B	A	B	A	B
P1	3	0	5	6	2	6	6	5
P2	2	2	9	5	7	3		
P3	1	3	9	8	8	5		
P4	0	5	8	5	8	0		

Sequence:

- ❑ **P1 can't go** because $\text{Need.B}(6) > \text{Available.B}(5)$
 - ❑ **P2 can't go** because $\text{Need.A}(7) > \text{Available.A}(6)$
 - ❑ **P3 can't go** because $\text{Need.A}(8) > \text{Available.A}(6)$
 - ❑ **P4 can't go** because $\text{Need.A}(8) > \text{Available.A}(6)$
- ❑ Hence, there is no safe sequence and this is snapshot is in “unsafe” state

Deadlock Detection

Detection: in a system that allows the possibility of deadlock, determine if deadlock has occurred, and which processes and resources are involved

Check to see if a deadlock has occurred!

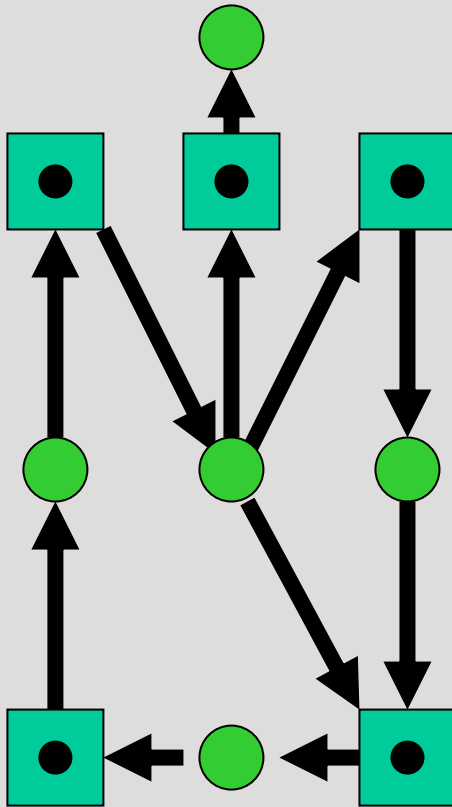
Single resource per type

Can use wait-for graph

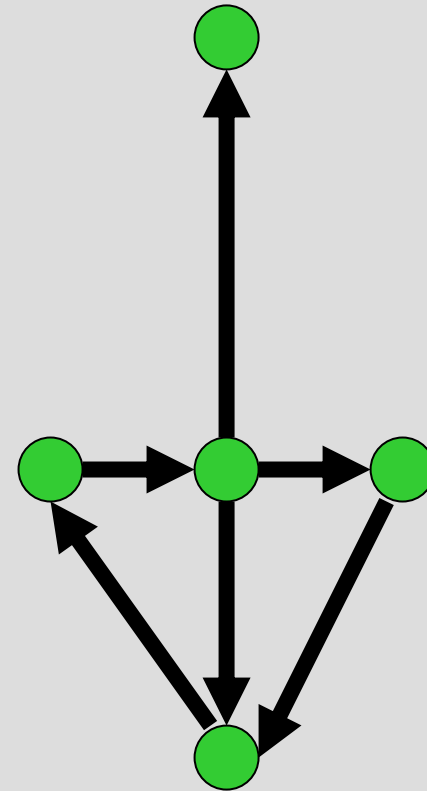
Check for cycles

$O(n^2)$

Wait for Graphs



Resource Allocation Graph



Corresponding Wait For Graph

Deadlock Recovery

Recovery: after a deadlock has been detected, clear the problem, allowing the deadlocked processes to complete and the resources to be reused

How? (Hint: traffic deadlock)

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: with rollback, difficult to prevent indefinite postponement

The Ostrich Algorithm

Guess: what is implemented in Linux?

Don't do
your
probl
Rationa
make
Dead
alg
if dea
ov



system (stick
there is no
e reliable
ection/recovery
worth the
e algorithms.

Summary

Deadlock issues

Deadlock Prevention - Havendar's Algorithms

Deadlock Avoidance – Banker Algorithm

Deadlock Detection – Wait-For-Graphs

Deadlock Recovery