

Operating Systems

Deadlock

Yanyan Zhuang

Department of Computer Science

<http://www.cs.uccs.edu/~yzhuang>

Recap of Last Class

- Race conditions
- Mutual exclusion and critical regions
- Busy waiting
 - Strict alternation and TSL
- Sleep/Wakeup
 - Semaphores
 - Mutexes

Resource Acquisition

```
typedef int semaphore;  
semaphore resource_1;
```

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

```
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);  
}
```

Using semaphore to protect resources. (a) One resource. (b) Two resources.

Resource Acquisition (2)

```
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) Deadlock-free code.



Resource Acquisition (2)

```
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) Deadlock-free code.

```
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_2);
    down(&resource_1);
    use_both_resources();
    up(&resource_1);
    up(&resource_2);
}
```

(b) Code with a potential deadlock, **why?**



Deadlock Definitions

- Two or more processes each blocked and waiting for resources they will never get without drastic actions
 - Something preempts a resource
 - A process is killed
- 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, thus, no process can
 - Run
 - Release resources
 - Be awakened

Resources and Deadlocks (1)

- Deadlocks occur when ...
 - Processes are granted *exclusive access* to hardware, e.g., I/O devices
 - Processes are granted *exclusive access* to software, e.g., database records
 - We refer to these generally as resources
- Pre-emptible resources
 - Can be taken away from a process with no ill effects, e.g., memory
- Non-preemptible resources
 - Will cause the process to fail if taken away, e.g., CD burner

In general, deadlocks involve *non-preemptible* and *exclusive* resources!



Resources and Deadlocks (2)

- Sequence of events required to use a resource
 - Request the resource
 - Use the resource
 - Release the resource
- Must **wait** if request is denied
 - Requesting process will be blocked

Four Conditions for Deadlock

Coffman (1971)

1. Mutual exclusion condition

- Each resource assigned to exactly 1 process (i.e., unavailable) or is available

2. Hold and wait condition

- Process holding resources request additional

3. No preemption condition

- Previously granted resources cannot be forcibly taken away

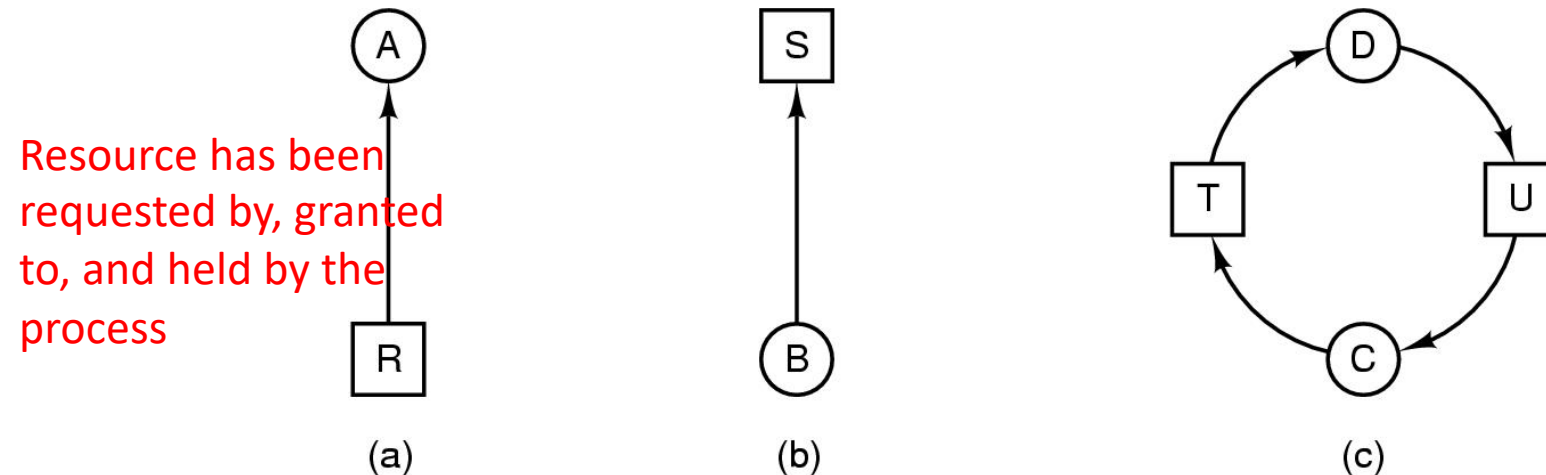
4. Circular wait condition

- Must be a circular chain of 2 or more processes
- Each is waiting for resources held by next member of the chain



Deadlock Modeling (1)

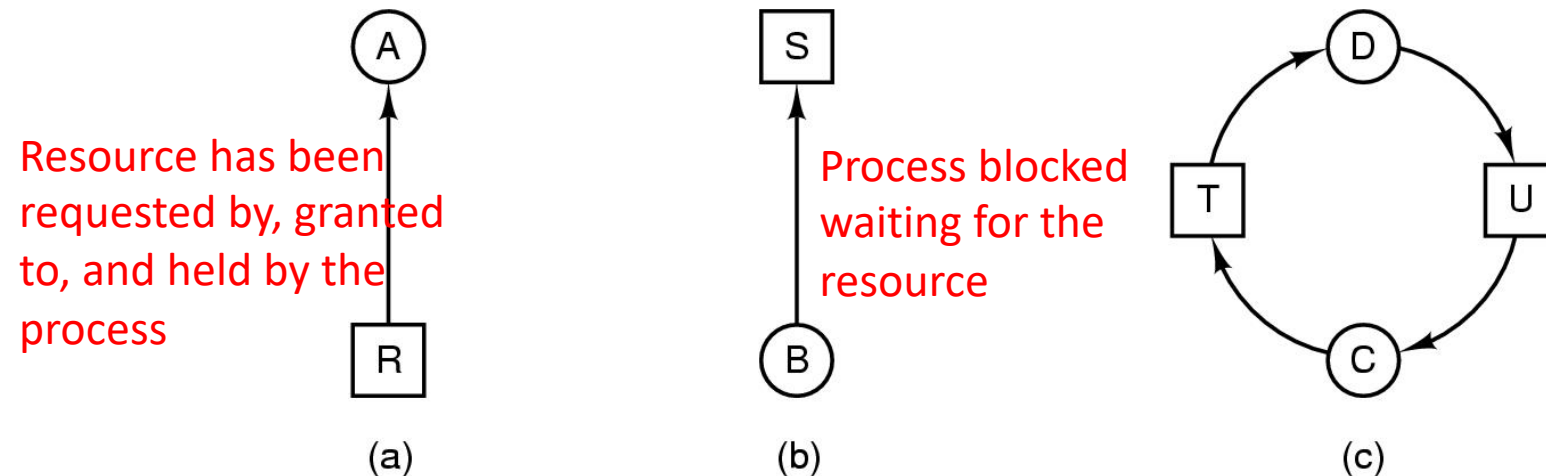
- Modeled with directed graphs: resource graphs
 - Processes (circles), resources (squares)
 - A cycle means a deadlock involving the processes and resources



- a) Resource R assigned to process A
- b) Process B is requesting/waiting/blocked for resource S
- c) Process C and D are in deadlock over resources T and U

Deadlock Modeling (1)

- Modeled with directed graphs: resource graphs
 - Processes (circles), resources (squares)
 - A cycle means a deadlock involving the processes and resources

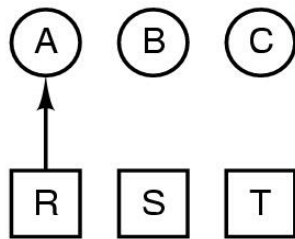


- a) Resource R assigned to process A
- b) Process B is requesting/waiting/blocked for resource S
- c) Process C and D are in deadlock over resources T and U

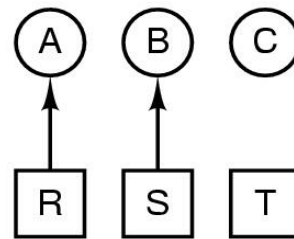
Deadlock Modeling (2)

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

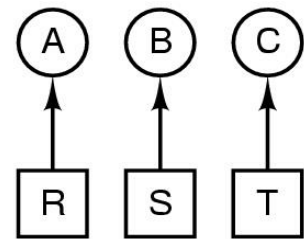
(d)



(e)

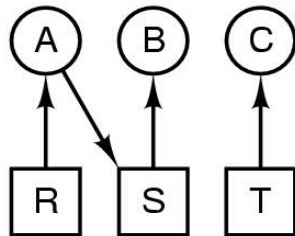


(f)

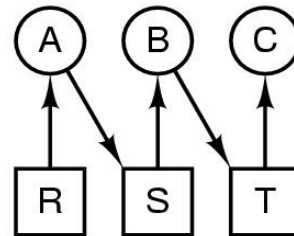


(g)

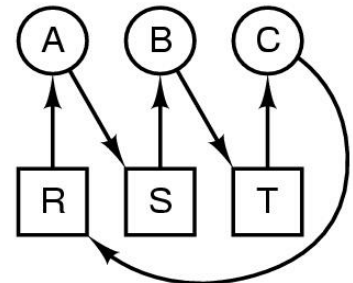
What if the OS knew the impending deadlock?



(h)



(i)

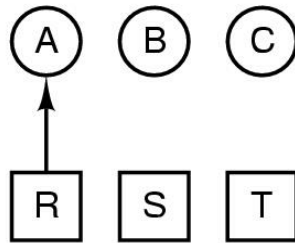


(j)

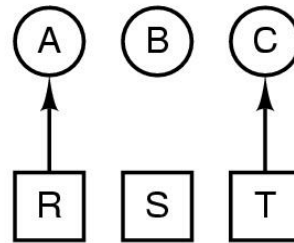
Deadlock Modeling (3)

1. A requests R
2. C requests T
3. A requests S
4. C requests R
5. A releases R
6. A releases S
no deadlock

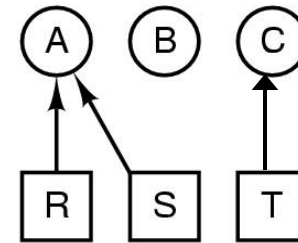
(k)



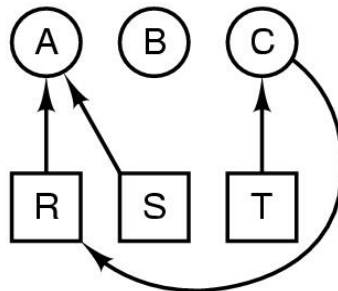
(l)



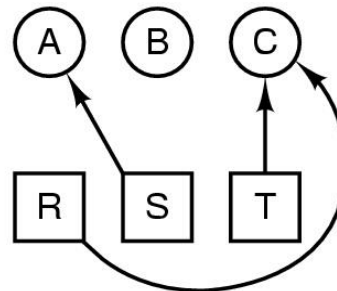
(m)



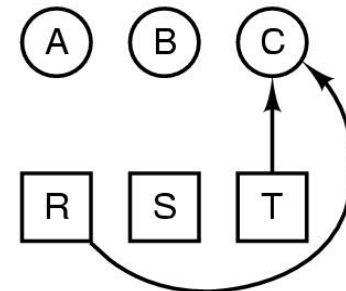
(n)



(o)



(p)



(q)

How deadlock can be avoided by OS' re-ordering (delaying B's requests)



Dealing with Deadlocks

- Strategies for dealing with Deadlocks

1. Just ignore the problem altogether

- ▶ Maybe if you ignore it, it will ignore you

2. Detection and recovery

- ▶ Let them occur, detect them, and take action

3. Dynamic avoidance

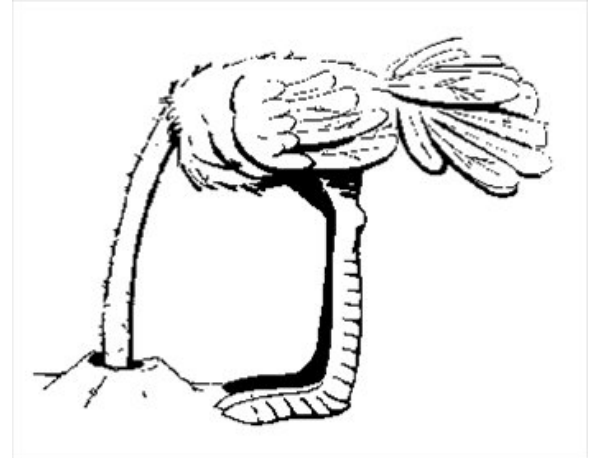
- ▶ By careful resource allocation

4. Prevention

- ▶ By structurally negating one of the four conditions

The Ostrich Algorithm

- Pretend there is no problem
- Reasonable if
 - Deadlocks occur very rarely
 - Cost of prevention is high
- UNIX and Windows take this approach
- It is a trade off between
 - Convenience
 - Correctness



Detection and Recovery

- Not attempt to prevent deadlocks
 - Let deadlocks occur
 - Tries to detect when it happens
 - Takes some action to recover after the fact

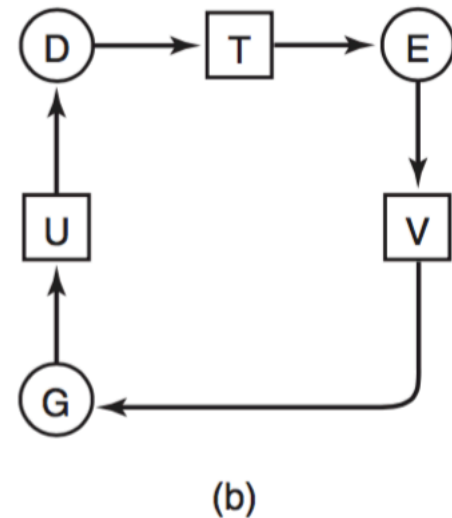
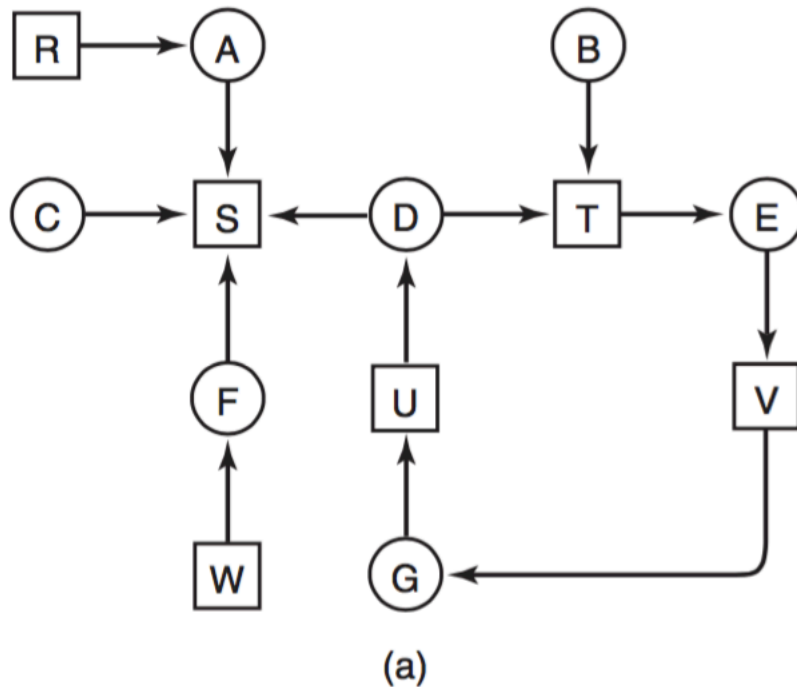
Detection and Recovery

- Not attempt to prevent deadlocks
 - Let deadlocks occur
 - Tries to detect when it happens
 - Takes some action to recover after the fact
- Detection & recovery: example – **one** resource of each type
 - Process A holds R and wants S.
 - Process B holds nothing but wants T.
 - Process C holds nothing but wants S.
 - Process D holds U and wants S and T.
 - Process E holds T and wants V.
 - Process F holds W and wants S.
 - Process G holds V and wants U.



Detection with One Resource of Each Type

- Assumption: **only one resource of each type exists**

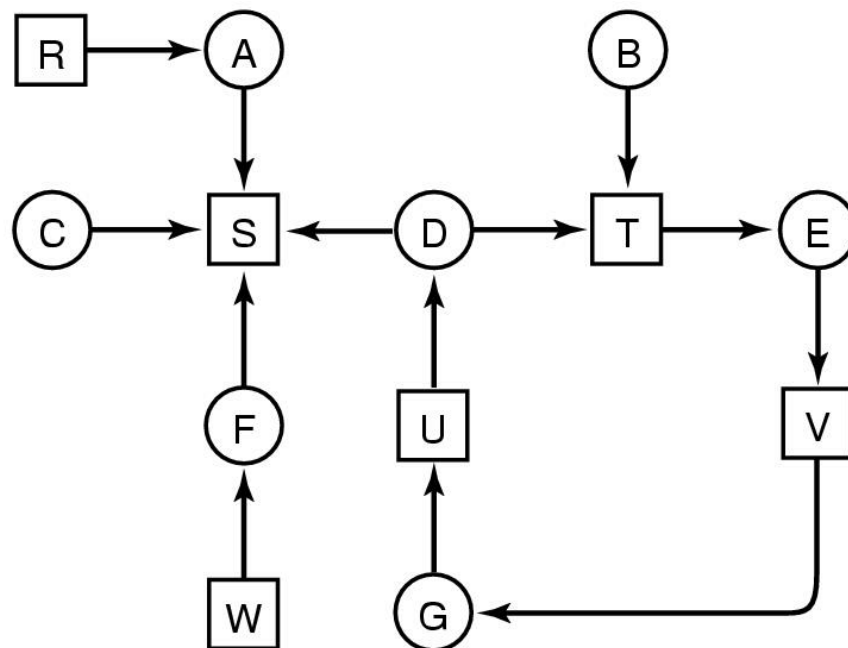


- Note the resource ownership and requests
- A cycle can be found within the graph, denoting deadlock

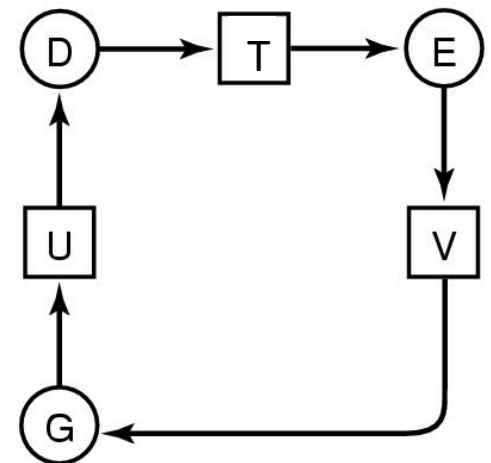


Detect a Cycle in a Graph

- A data structure to find if a graph is a tree that is cycle-free
 - Take a node as root, do depth-first search (P.445)
 - Left-right, top-to-bottom: R, A, B, C, S, D, T, E, F



(a)



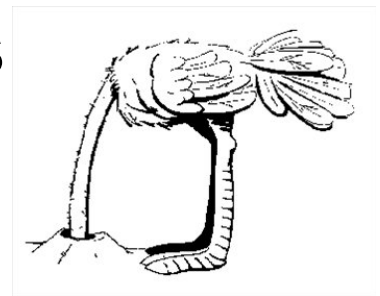
(b)

Recap: Dealing with Deadlocks

- Strategies for dealing with Deadlocks

1. Just ignore the problem altogether

- ▶ Maybe if you ignore it, it will ignore you



2. Detection and recovery

**One resource of each type:
resource graph + cycle detection**

- ▶ Let them occur, detect them, and take action

3. Dynamic avoidance

**Multiple resource of each type:
vectors + matrices**

- ▶ By careful resource allocation

4. Prevention

- ▶ By structurally negating one of the four conditions

Detection with Multiple Resources of Each Type (1)

- Deadlock detection algorithm:
 - Two vectors and two matrixes: n processes, m resource types
 - Vector comparison; $X \leq Y$ means $X_i \leq Y_i$ for $1 \leq i \leq m$
 - Observation:

Resources in existence

$(E_1, E_2, E_3, \dots, E_m)$

At any instant, $A \leq E$

Resources available

$(A_1, A_2, A_3, \dots, A_m)$

Detection with Multiple Resources of Each Type (1)

- Deadlock detection algorithm:

- Two vectors and two matrixes: n processes, m resource types
- Vector comparison; $X \leq Y$ means $X_i \leq Y_i$ for $1 \leq i \leq m$
- Observation:

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

Resources in existence

$(E_1, E_2, E_3, \dots, E_m)$

At any instant, $A \leq E$

Resources available

$(A_1, A_2, A_3, \dots, A_m)$

Current allocation matrix

$$\begin{bmatrix} C_{11} & C_{12} & C_{13} & \cdots & C_{1m} \\ C_{21} & C_{22} & C_{23} & \cdots & C_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ C_{n1} & C_{n2} & C_{n3} & \cdots & C_{nm} \end{bmatrix}$$

Row n is current allocation
to process n

Request matrix

$$\begin{bmatrix} R_{11} & R_{12} & R_{13} & \cdots & R_{1m} \\ R_{21} & R_{22} & R_{23} & \cdots & R_{2m} \\ \vdots & \vdots & \vdots & & \vdots \\ R_{n1} & R_{n2} & R_{n3} & \cdots & R_{nm} \end{bmatrix}$$

Row 2 is what process 2 needs

Data structures needed by deadlock detection algorithm



Detection with Multiple Resources of Each Type (2)

- Key: a completed process can release its resources so as to give other processes chances to acquire resources and run
 - Look for a process P_i , If $R[i] \leq A$? if so, **$A = A + C[i]$ and repeat** until a scheduling order is found; if no order can be found, deadlock
 - $A = A - R[i] + C[i] + R[i]$

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives
Plotters
Scanners
CD Roms

An example for the deadlock detection algorithm

Current allocation matrix

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

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} P1 \\ P2 \\ P3 \end{matrix}$$

Detection with Multiple Resources of Each Type (2)

- Key: a completed process can release its resources so as to give other processes chances to acquire resources and run
 - Look for a process P_i , If $R[i] \leq A$? if so, **$A = A + C[i]$** and **repeat** until a scheduling order is found; if no order can be found, deadlock

Though order is non-deterministic,
result (deadlock or not) is the same

$$E = \begin{pmatrix} 4 & 2 & 3 & 1 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

$$A = \begin{pmatrix} 2 & 1 & 0 & 0 \end{pmatrix}$$

Tape drives Plotters Scanners CD Roms

What if process 3 needs a CD-ROM drive in addition to 2 tape drivers and 1 plotter?
When to run the deadlock detection algorithm?

Current allocation matrix

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

Request matrix

$$R = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix} \begin{matrix} P1 \\ P2 \\ P3 \end{matrix}$$

An example for the deadlock detection algorithm



When to Detect Deadlock

- Check every time a resource request is made
 - Detect as early as possible
 - Potentially expensive in terms of CPU time
- Check every k minutes
- When CPU utilization has dropped below some threshold
 - Enough processes are deadlocked → few runnable processes → CPU will often be idle

Recovery from Deadlock (not attractive)

- Recovery through preemption
 - take a resource from some other process
 - depends on the nature of the resource (frequently difficult or impossible)

Recovery from Deadlock (not attractive)

- Recovery through preemption
 - take a resource from some other process
 - depends on the nature of the resource (frequently difficult or impossible)
- Recovery through rollback
 - *checkpoint* a process periodically, resulting a sequence of checkpoint files
 - use this saved state
 - restart the process if it is found deadlocked

Recovery from Deadlock (not attractive)

- Recovery through preemption
 - take a resource from some other process
 - depends on the nature of the resource (frequently difficult or impossible)
- Recovery through rollback
 - *checkpoint* a process periodically, resulting a sequence of checkpoint files
 - use this saved state
 - restart the process if it is found deadlocked
- Recovery through killing processes
 - crudest but simplest way to break a deadlock
 - kill one of the processes in the deadlock cycle
 - the other processes get its resources
 - choose process that can be rerun from the beginning, **not easy!**

Dealing with Deadlocks

- Strategies for dealing with Deadlocks

1. Just ignore the problem altogether

- ▶ Maybe if you ignore it, it will ignore you

2. Detection and recovery

- ▶ Let them occur, detect them, and take action

3. Dynamic avoidance

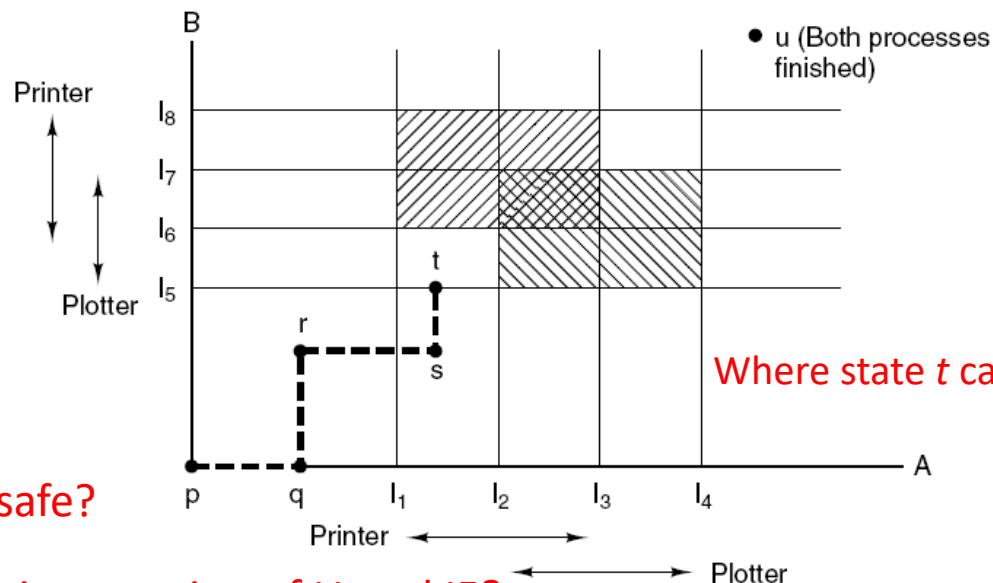
- ▶ By careful resource allocation

4. Prevention

- ▶ By structurally negating one of the four conditions

Deadlock Avoidance

- Allocate resources wisely to avoid deadlocks
 - Resources are requested one at a time, not all at once (R matrix)
 - But certain information should be available in advance
 - Base: resource trajectories, concept of safe states



Where state *t* can go to avoid deadlock?

Which area is unsafe?

What if *t* is at the intersection of *l*₁ and *l*₅?

Two process resource trajectories (one CPU)

Safe States (w/ one resource type)

- Safe state
 - If it is not deadlocked, and, there is *some* scheduling order in which every process can run to completion even **if all of them request their maximum** number of resources immediately

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

(a)

Why state (a) is safe? By careful scheduling



Safe States (w/ one resource type)

- Safe state

- If it is not deadlocked, and, there is *some* scheduling order in which every process can run to completion even **if all of them request their maximum** number of resources immediately

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)

Why state (a) is safe? By careful scheduling



Unsafe States (w/ one resource type)

- Unsafe state
 - There is *no guarantee* of having some scheduling order in which every process can run to completion even **if all of them request their maximum** number of resources immediately
 - Not the same as a deadlocked state, **why? What is the difference?**

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)

Why state (b) is NOT safe?



Unsafe States (w/ one resource type)

- Unsafe state
 - There is *no guarantee* of having some scheduling order in which every process can run to completion even **if all of them request their maximum** number of resources immediately
 - Not the same as a deadlocked state, **why? What is the difference?**

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	—	—
C	2	7

Free: 4

(d)

Why state (b) is NOT safe?



The Banker's Algorithm for a Single Resource

- The algorithm models on the way of a banker might deal with a group of customers to whom he has granted lines of credit
 - Not all customers need their maximum credit line simultaneously
 - To see if a state is safe, the banker checks to see if he has enough resources to satisfy some customer

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)

Three resource allocation states: (a) safe; (b) safe; (c) unsafe



The Banker's Algorithm for Multiple Resources (1)

- The algorithm looks for a process P_i , If $R[i] \leq A$? if so, $A = A + C[i]$
 - Given current allocation (C) and maximum needs (M): $R = M - C$
 - What is the underlying assumption? M info available in advance

C

	Process	Tape drives	Plotters	Scanners	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

R

	Process	Tape drives	Plotters	Scanners	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

Existing
 Possessed
 Available
 $E = P + A$

Can process B's request for a scanner be granted? Why or why not?

The Banker's Algorithm for Multiple Resources (2)

	Process	Tape drives	Plotters	Scanners	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	Scanners	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)

After process B was granted a scanner, now process E wants the last scanner, can it be granted? Why?

Dealing with Deadlocks

- Strategies for dealing with Deadlocks

1. Just ignore the problem altogether

- ▶ Maybe if you ignore it, it will ignore you

2. Detection and recovery

- ▶ Let them occur, detect them, and take action

3. Dynamic avoidance

- ▶ By careful resource allocation

4. Prevention

- ▶ By structurally negating one of the four conditions

Four Conditions for Deadlock

Coffman (1971)

1. Mutual exclusion condition

- Each resource assigned to exactly 1 process or is available

2. Hold and wait condition

- Process holding resources request additional

3. No preemption condition

- Previously granted resources cannot be forcibly taken away

4. Circular wait condition

- Must be a circular chain of 2 or more processes
- Each is waiting for resources held by next member of the chain



Deadlock Prevention (1)

Attack the mutual exclusion condition of Coffman Rules

- No resource is ever exclusively assigned to a process
- Some devices (such as printer) can be spooled
 - Only the printer daemon uses printer resource
 - Thus deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
 - Avoid assigning resource unless absolutely necessary
 - As few processes as possible actually claim the resource

Deadlock Prevention (2)

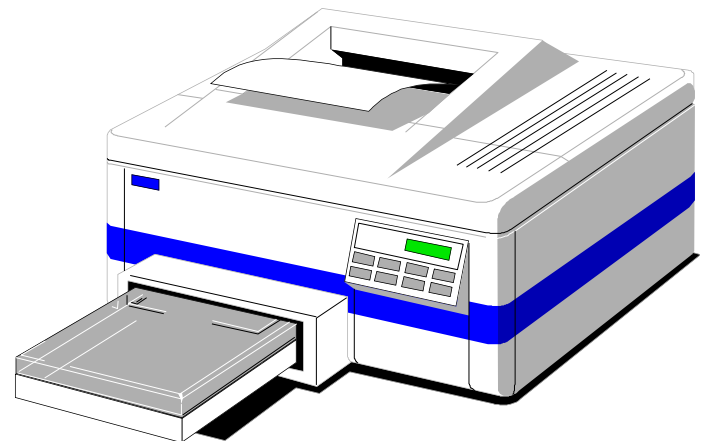
Attack the Hold-and-Wait condition of Coffman Rules

- Require processes to request *all* resources before starting
 - A process never has to wait for what it needs if all available
- Problems
 - May not know required resources at start of run
 - ▶ Otherwise banker's algorithm can be used
 - Also ties up resources other processes could be using
 - ▶ **Less concurrency!**
- Variation:
 - Process must temporarily give up all resources
 - Then request all immediately needed

Deadlock Prevention (3)

Attack the No-Preemption Condition of Coffman Rules

- This is not a viable option
- Consider a process given the printer
 - Halfway through its job
 - Now forcibly take away printer
 - !!??



Deadlock Prevention (4)

Attack the Circular Wait Condition of Coffman Rules

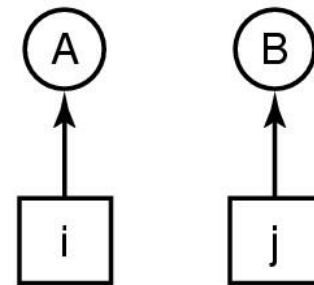
- A process is entitled only to a single resource at any moment
 - If it needs a second one, it must release the first one
- Provide a global numbering of all the resources
 - A process can request resources whenever they want to, but all requests must be made in numerical order (or no process requests a resource lower than what it is already holding)

- Why no deadlock?
- Is it feasible in implementation? – what is a good ordering?

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

(a) Normally ordered resources

$i > j$: A is not allowed to request j ;
 $i < j$: B is not allowed to request i



(b) a resource graph

Deadlock Prevention Summary

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

Summary of approaches to deadlock prevention

What is the difference between deadlock avoidance and deadlock prevention?

dynamic scheduling vs. static ruling



Starvation

- Some algorithm to allocate a resource
 - May be to give to shortest job first
 - Works great for multiple short jobs in a system
- It may cause long jobs to be postponed indefinitely
- Solution:
 - First-come, first-serve
 - Round robin

What is the key difference between deadlock and starvation?

Deadlock: two or more processes are unable to proceed because each is waiting for one the others to do something.

Starvation: A situation in which a runnable process is overlooked indefinitely by the scheduler; although it is able to proceed, it is never chosen.

Summary

- Deadlocks and its modeling
- Deadlock detection
- Deadlock recovery
- Deadlock avoidance
 - Resource trajectories
 - Safe and unsafe states
 - The banker's algorithm