Operating Systems

Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/

Deadlock

- When processes try to acquire resources concurrently they may end up “stuck”
- Process A needs P, Q
- Process B needs Q, P
- Process A gets P
- Process B gets Q
- Process A tries to get Q and blocks
- Process B tries to get P and blocks

Resources

- Examples of computer resources
  - Printers
  - Tape drives
  - Tables
- Resources can be available
  - In a single instance (e.g., one printer)
  - In multiple identical copies (e.g., an array of tape drives)
- Resources can be
  - Preemptable: the resource can be taken away from a process with no negative side-effects
  - Nonpreemptable: taking away the resource will cause the process to fail

Accessing Resources

- Deadlocks occur when processes are granted exclusive access to non-preemptable resources
- Sequence of events required to use a resource
  - Request the resource
  - Use the resource
  - Release the resource
- If request is denied
  - Requesting process may be blocked
  - May fail with error code

Defining Deadlocks

- Formal definition :
  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

- Usually the event is the release of a currently held resource

- None of the processes can
  - Run
  - Release resources
  - Be awakened

Four Conditions for Deadlock

1. Mutual exclusion condition

   Each resource is assigned to exactly one process or is available

2. Hold and wait condition

   A process holding resources can request additional ones

3. No preemption condition

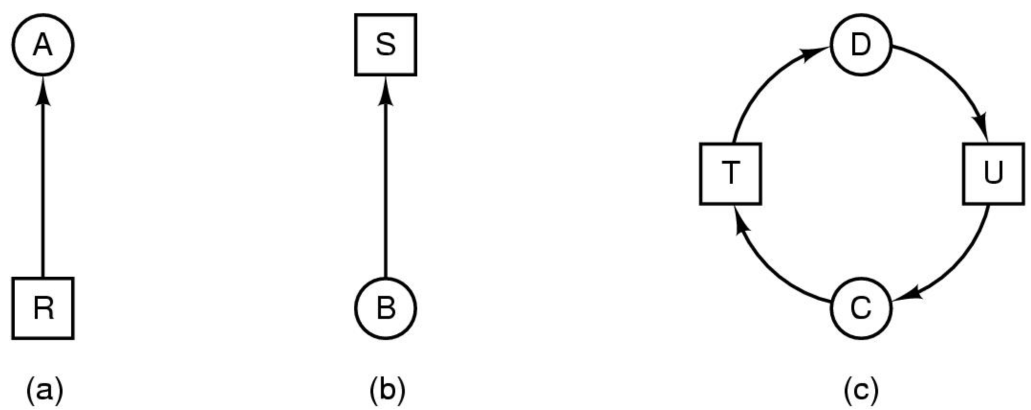
   Previously granted resources cannot forcibly be taken away

4. Circular wait condition

   There must be a circular chain of two or more processes, each of which is waiting for a resource held by next member of the chain

Deadlock Modeling

- Modeled with directed graphs
  - Processes: circles
  - Resources: squares



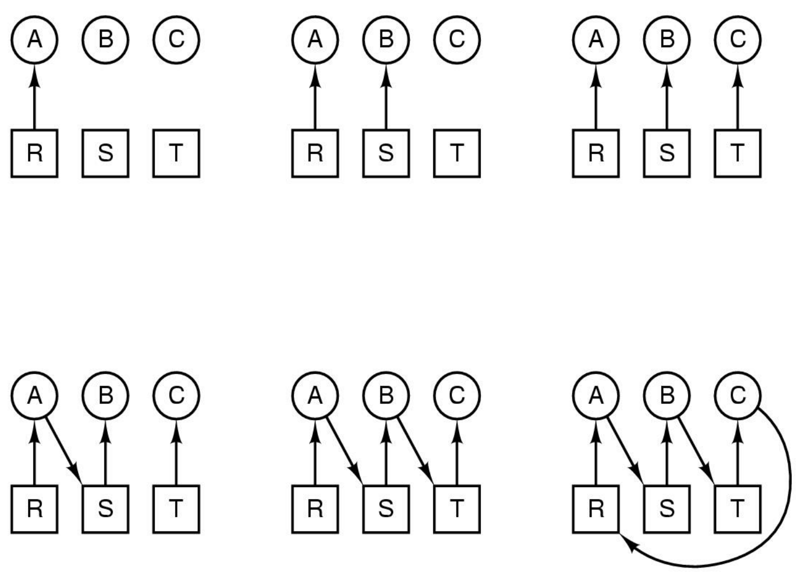
(a) Resource R assigned to process A
(b) Process B is requesting/waiting for resource S
(c) Process C and D are in deadlock over resources T and U

An Example

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R

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

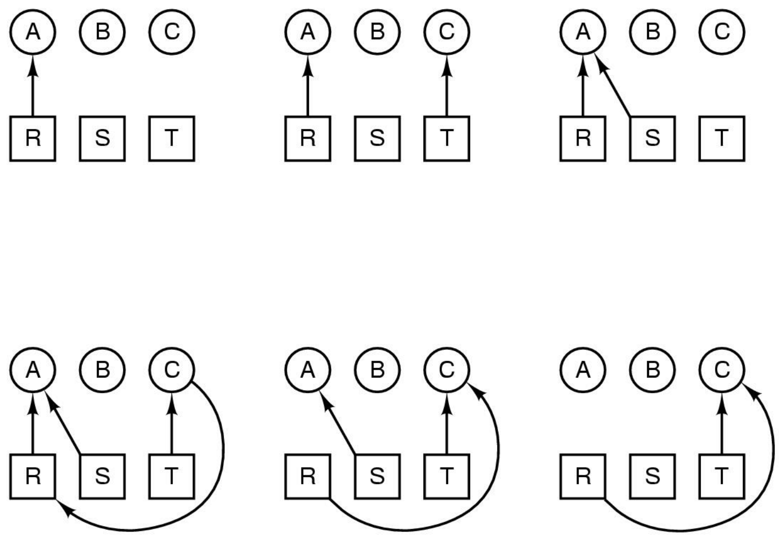
   deadlock



Another Example

A	B	C
Request R	Request S	Request T
Request S	Request T	Request R
Release R	Release S	Release T
Release S	Release T	Release R

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



Dealing With Deadlocks

- Just ignore the problem altogether
  - Bad things happen!
- Detection and recovery
  - Let them occur and deal with it
- Dynamic avoidance
  - Careful resource allocation
- Prevention
  - Negating one of the four necessary conditions

The Ostrich Algorithm

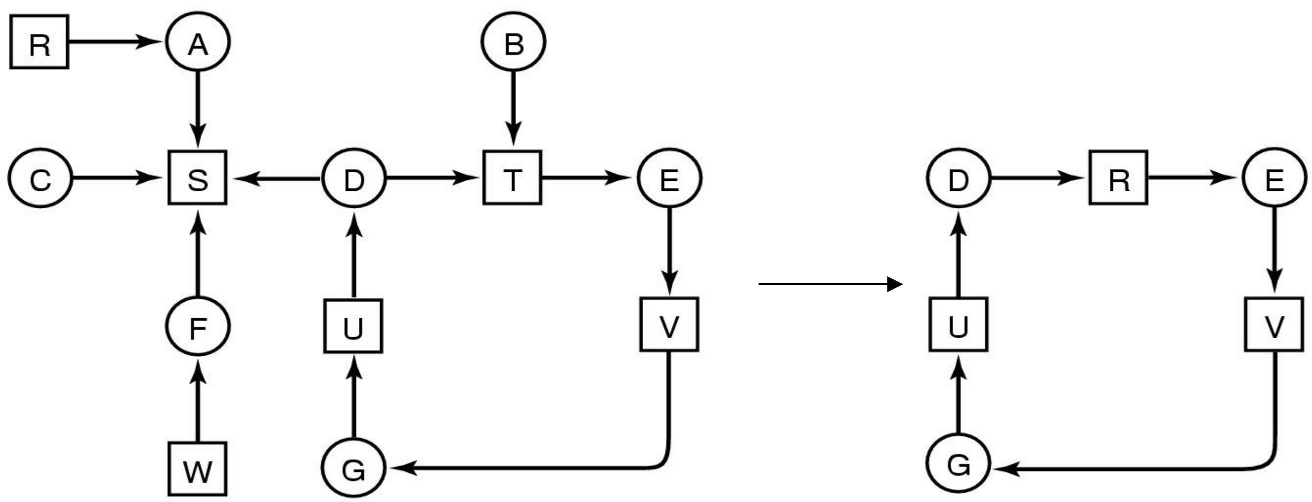
- Pretend there is no problem
- Reasonable if
  - Deadlocks occur very rarely
  - Cost of prevention is high
- UNIX and Windows takes this approach
- It is a trade off between
  - Convenience
  - Correctness

Detection And Recovery

- Let deadlocks happen and deal with the situation
- Need to detect: Deadlock detection algorithms
- Need to recover: Preemption, Rollback, Killing

Detection with One Resource of Each Type

- Note the resource ownership and requests
- If a cycle can be found within the graph, then there is a deadlock



Detection with One Resource of Each Type

L: list of nodes

Arcs can be marked to indicate that they have been inspected

1. For each node N in the graphs do the following
2. L := empty, arcs all unmarked
3. Add current node to L and check if it appears two times
   1. Yes: there is a cycle
   2. No: continue
4. Are there outgoing, unmarked arcs? If not go to step 6
5. Pick randomly an unmarked arc and mark it, follow the arc to the node and go to step 3
6. Remove current node from the list, go back to the previous node, and jump to step 3. If this is the root node then there are no cycles

Detection with Multiple Resources of Each Type

Resources in existence
$(E_1, E_2, E_3, \ldots, E_m)$

Resources available
$(A_1, A_2, A_3, \ldots, A_m)$

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

Detection with Multiple Resources of Each Type

- Comparing vectors: $A < B$ iff for every corresponding element A_i, B_i it is $A_i < B_i$

- Initially all processes are unmarked (not deadlocked)

- Look for a process for which the corresponding row in R is less than or equal to A
  - If such process exists add the corresponding row of C to A, mark the process and restart to look
  - If there is no such process then exit

- At the end, unmarked processes are in deadlock

Detection with Multiple Resources of Each Type

	Tape drives	Plotters	Scanners	CD Roms
E =	4	2	3	1

	Tape drives	Plotters	Scanners	CD Roms
A =	2	1	0	0

$$E = (4 \quad 2 \quad 3 \quad 1) \qquad A = (2 \quad 1 \quad 0 \quad 0)$$

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
  - Take a resource from some other process
  - Depends on nature of the resource

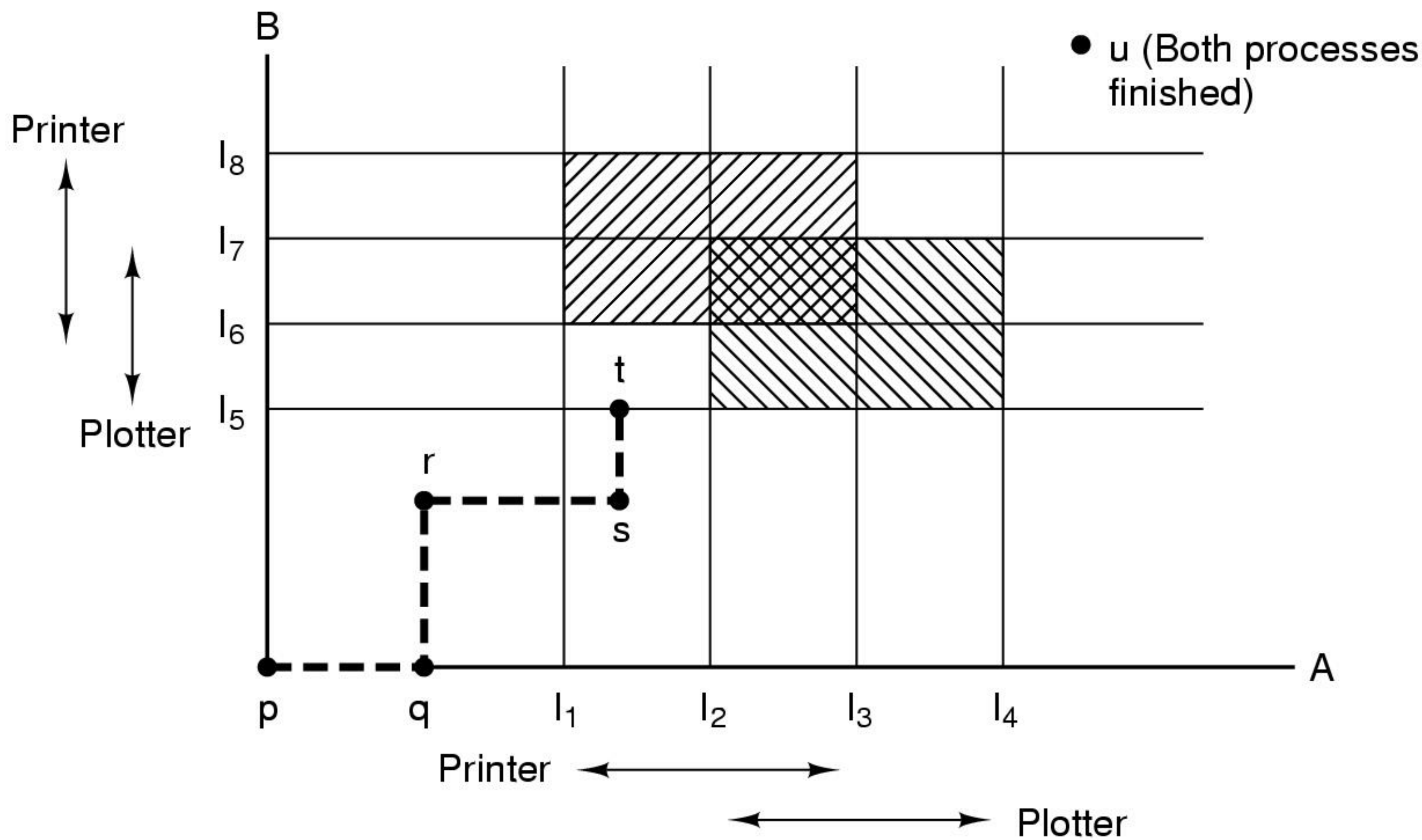
- Recovery through rollback
  - Checkpoint a process periodically
  - Use this saved state
  - Restart the process if it is found deadlocked

- 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

Deadlock Avoidance

- When a process requests a resource the system must decide if resource should be granted
- To avoid deadlocks system should stay in *safe state*
- State: matrices C, R, E, A
- Safe state: there is currently no deadlocked process and there is some scheduling order in which every process can run to completion, even if all the processes request all the resources at the same time

Resource Trajectories



Safe and Unsafe States

- State (a) is safe
  - Max possible allocation is A=6, B=2, C=5
  - Three are free, give two to B and let it run to completion
- State (c)
  - Max possible allocation is A=6, C=5
  - Five are free give to C and let it run to completion

	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)

Total number of resources: 10

Moving to an Unsafe State

- From State (a) one resource is given to A
- State (b) is unsafe because there is not a scheduling order in which every process can run to completion if all the processes request all the resources at the same time
- B gets 2 and returns 4, but both A and C need 5

	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)

The Banker's Algorithm

- Algorithm considers each request and examines if it leads to a safe state
  - Check if there are enough resources to satisfy at least one process
  - Sum the resources of the process to those available, mark the process and iterate
  - If at the end there are processes that are left unmarked the process would lead to an unsafe state

- If granting the request would lead to an unsafe state then resource is not granted

The Banker's Algorithm

- B requests a scanner. Should the scanner be granted?
- Then E requests the last scanner. Should it be granted?

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)

Deadlock Prevention

Invalidate one of the following:

1. Mutual exclusion condition

   Each resource is assigned to exactly one process or is available

2. Hold and wait condition

   A process holding resources can request additional ones

3. No preemption condition

   Previously granted resources cannot forcibly be taken away

4. Circular wait condition

   There must be a circular chain of two or more processes, each of which is waiting for a resource held by next member of the chain

Attacking the Mutual Exclusion Condition

- Some devices (such as printer) can be spooled
  - Only the printer daemon uses printer resource
  - Deadlock for printer eliminated
- Not all devices can be spooled
- Principle:
  - Avoid assigning resource when not absolutely necessary
  - As few processes as possible actually claim the resource

Attacking the Hold and Wait Condition

- Require processes to request all their resources before starting
  - A process never has to wait for what it needs

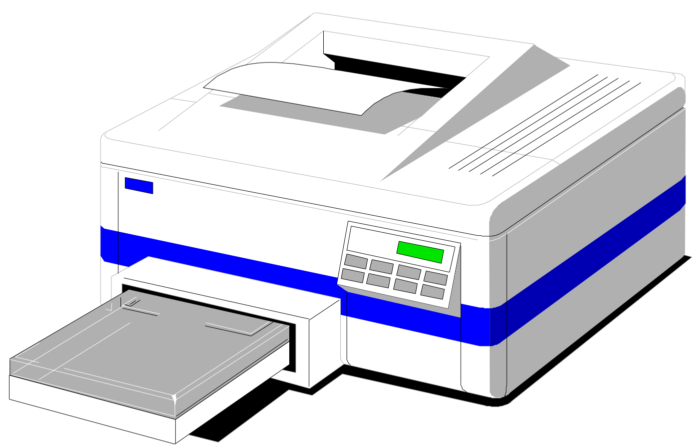
- Problems
  - Process may not know required resources at start of run
  - Ties up resources that other processes could be using

- Possible solution
  - Process must give up all resources before acquiring a new one
  - Then request all needed resources at once

Attacking the No Preemption Condition

- This is not a very appealing option

- Consider a process that is using a printer
  - Let process go halfway through its job
  - Then forcibly take away printer
  - Results can be unpredictable



Attacking the Circular Wait Condition

- Require a process to request/hold only one resource at a time
- Provide global numbering of resources and require ordered acquisitions
  - A process holding resource *j* cannot ask for resource *i*, with $i < j$
- The resulting resource graph is cycle-free

Two-Phase Locking

- Phase One
  - Process tries to lock all records it needs, one at a time
  - If needed record found locked, release all the locks and start over

- If phase one succeeds, it starts second phase
  - Performing updates
  - Releasing locks

- Similar to requesting all resources at once

Non-resource Deadlocks

- Possible for two processes to deadlock
  - Each is waiting for the other to do some task

- Can happen with semaphores
  - Each process required to do a down() on two semaphores (mutex and another)
  - If done in wrong order, deadlock results

Starvation

- Algorithm to allocate a resource
  - May be to give to shortest job first

- Works great for multiple short jobs in a system

- May cause long job to be postponed indefinitely
  - Even though not blocked

- Solution:
  - First-come, first-serve policy