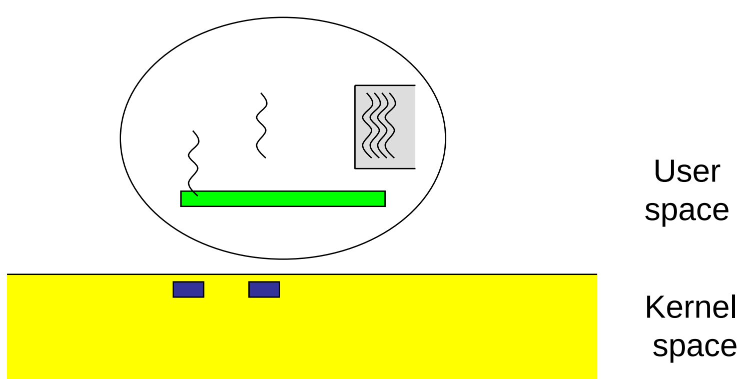
Deadlocks

Thomas Plagemann

With slides from C. Griwodz, K. Li,
A. Tanenbaum and M. van Steen

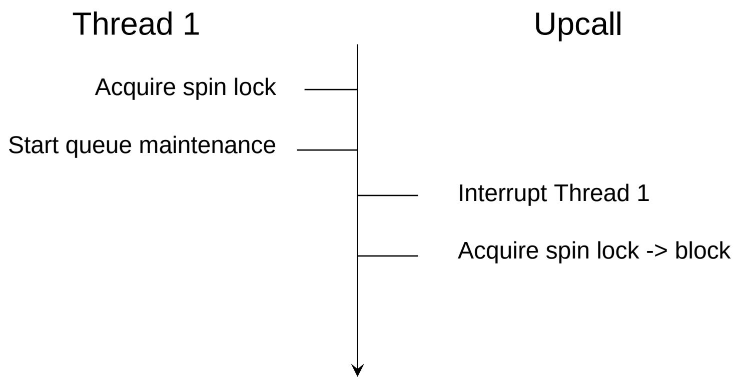
Preempting Scheduler Activations

- Scheduler activations are completely preemptable



Preempting Scheduler Activations

- Maintaining the run queue needs to be a protected critical section
- Let's use spin locks for protection



Resources

- Examples of computer resources
  - CPU
  - Memory
  - Disk drive
  - Tape drives
  - Printers
  - Plotter
  - Loudspeaker

Resources

- Processes
  - Need access to resources in reasonable order
- Typical way to use a resource
  - Request
  - Use
  - Release
- Suppose a process holds resource A and requests resource B
  - At same time another process holds B and requests A
  - Both are blocked and remain so

Resources

- Active resource
  - Provides a service
  - E.g. CPU, network adaptor
- Passive resource
  - System capabilities that are required by active resources
  - E.g. memory, network bandwidth
- Exclusive resource
  - Only one process at a time can use it
  - E.g. loudspeaker, processor
- Shared resource
  - Can be used by multiple processes
  - E.g. memory, bandwidth

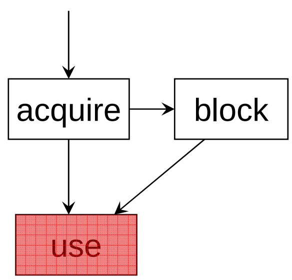
Resources

- Single resource
  - Exists only once in the system
  - E.g. loudspeaker
- Multiple resource
  - Exists several time in the system
  - E.g. processor in a multiprocessor system

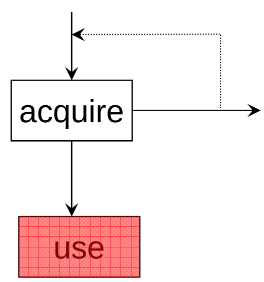
- Preemptable resource
  - Resource that can be taken away from a process
  - E.g. CPU can be taken away from processes in user space
- Non-preemptable resource
  - Taking it away will cause processes to fail
  - E.g. Disk, files

Resources

- Process must wait if request is denied
  - Requesting process may be blocked
  - May fail with error code



- Deadlocks
  - Occur only when processes are granted exclusive access to resources



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

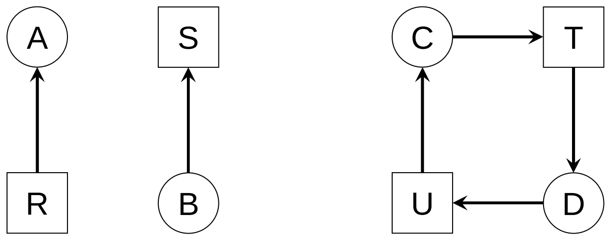
- Usually the *event* is release of a currently held resource
- None of the processes can …
  - Run
  - Release resources
  - Be awakened

Four Conditions for Deadlock

1. Mutual exclusion condition
   - Each resource assigned to 1 process or is available
2. Hold and wait condition
   - Process holding resources can request additional
3. No preemption condition
   - Previously granted resources cannot forcibly taken away
4. Circular wait condition
   - Must be a circular chain of 2 or more processes
   - Each is waiting for resource held by next member of the chain

Deadlock Modeling

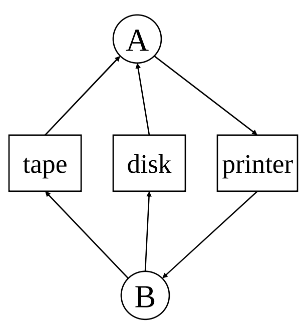
- Modeled with directed graphs



  - 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 Example

- A utility program
  - Copies a file from a tape to disk
  - Prints the file to a printer
- Resources
  - Tape
  - Disk
  - Printer
- A deadlock



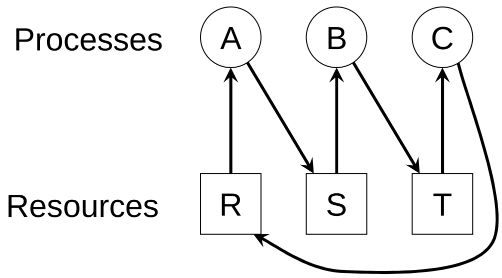
Deadlock Modeling

- How deadlock occurs

A
Requests R
Requests S
Releases S
Releases R

B
Requests S
Requests T
Releases T
Releases S

C
Requests T
Requests R
Releases R
Releases T



A requests R
B requests S
C requests T
A requests S
B requests T
C requests R

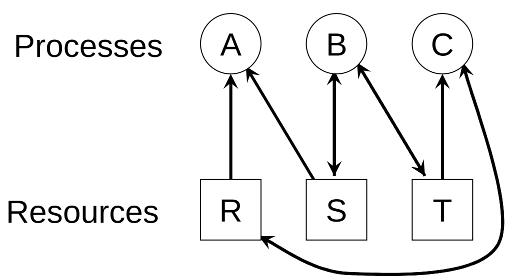
Deadlock Modeling

- How deadlock can be avoided

A
Requests R
Requests S
Releases S
Releases R

B
Requests S
Requests T
Releases T
Releases S

C
Requests T
Requests R
Releases R
Releases T



A requests R
C requests T
A requests S
B requests S
B requests T
C requests R
A releases S
A releases R
C releases R
C releases T

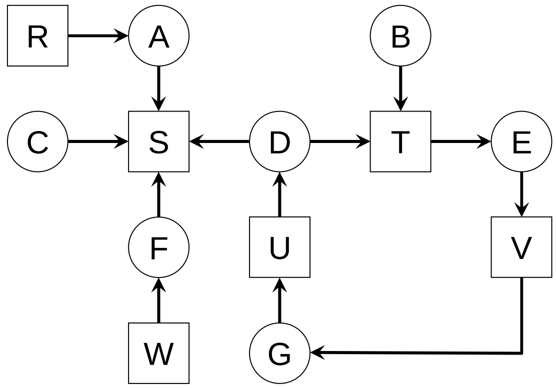
Deadlocks: Strategies

- Ignore the problem
  - It is user's fault
- Detection and recovery
  - Fix the problem afterwards
- Dynamic avoidance
  - Careful allocation
- Prevention
  - Negate one of the four conditions

The Ostrich Algorithm

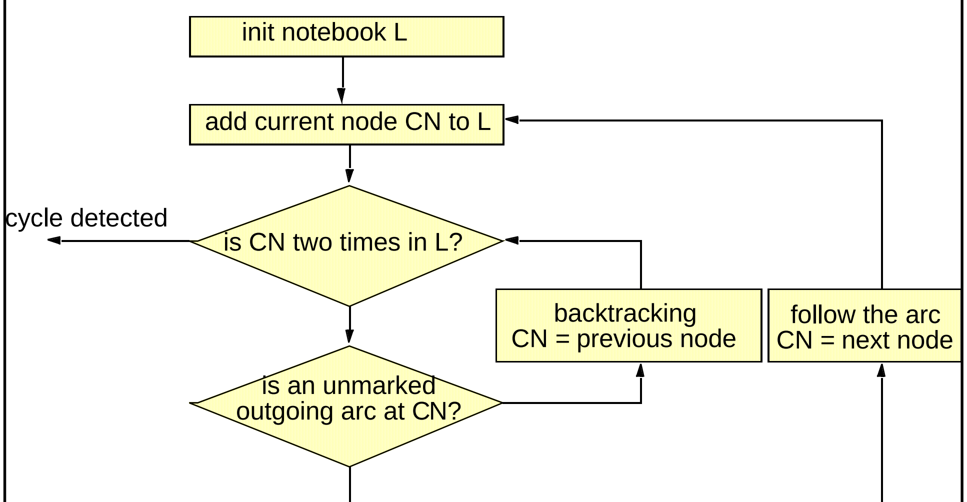
- Pretend there is no problem
- Reasonable if
  - Deadlocks occur very rarely
  - Cost of prevention is high
- UNIX and Windows take this approach
- It is a trade-off between
  - Convenience
  - Correctness

Deadlock Detection and Recovery
One Resource of Each Type



- A cycle can be found within the graph, denoting deadlock

Deadlock Detection and Recovery
One Resource of Each Type



Deadlock Detection and Recovery
Multiple Resources of Each Type

Existing resources

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

Available resources

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

Current allocation matrix

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

Request matrix

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

Deadlock Detection and Recovery
Multiple Resources of Each Type

	Tape drivers	Plotters	Scanners	CD-Roms
E=	4	2	3	1
A=	2	1	0	0

Current allocation matrix

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

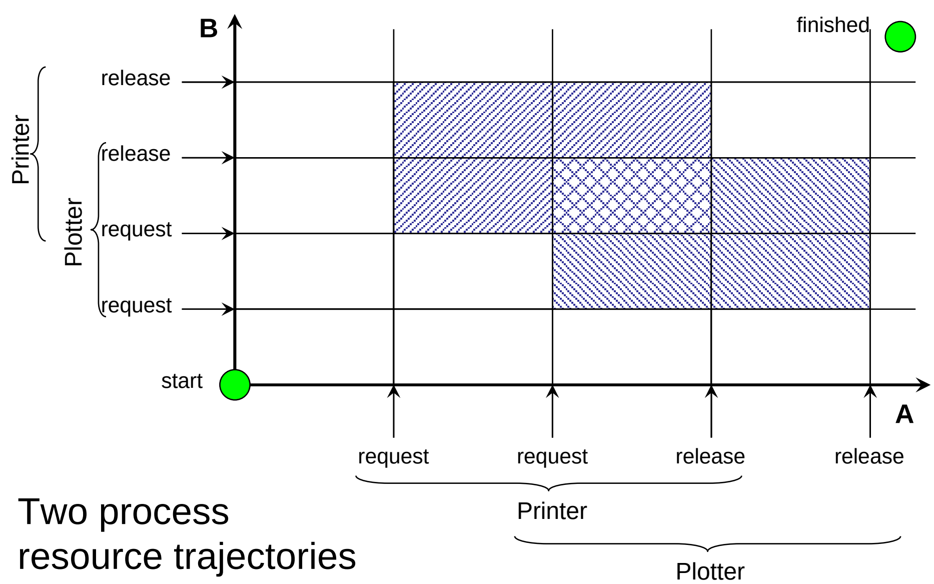
Request matrix

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

Deadlock Detection and Recovery
Recovery

- 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
Resource Trajectories



Two process
resource trajectories

Deadlock Avoidance
Safe and Unsafe States

	has	max
A	3	9
B	2	4
C	2	7

Free: 3

	has	max
A	3	9
B	4	4
C	2	7

Free: 1

	has	max
A	3	9
B	0	
C	2	7

Free: 5

	has	max
A	3	9
B	0	
C	7	7

Free: 0

	has	max
A	3	9
B	0	
C	0	

Free: 7

state is safe

Deadlock Avoidance
Safe and Unsafe States

	has	max
A	3	9
B	2	4
C	2	7

Free: 3

	has	max
A	4	9
B	2	4
C	2	7

Free: 2

	has	max
A	4	9
B	4	4
C	2	7

Free: 0

	has	max
A	3	9
B	0	
C	2	7

Free: 4

state is safe

Deadlock Avoidance
Banker’s Algorithm for a Single Resource

- Each process has a credit
  - System knows how many resources a process requests *at most* before releasing resources
- Total resources may not satisfy all credits
- Keep track of resources assigned and needed
- Check on each allocation whether it is safe
  - Safe: there exists a sequence of other states that all processes can terminate correctly

Deadlock Avoidance
Banker's Algorithm for a Single Resource

Resource allocation state

	has	max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

	has	max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

	has	max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	5	3	2	2
A=	1	0	2	0

Assigned resources

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 still needed

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

An example for the deadlock detection algorithm

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	4	2	2	1
A=	2	1	2	1

Assigned resources

A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	0	0	0	0
E	0	0	0	0

Resources still needed

A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	-	-	-	-
E	2	1	1	0

An example for the deadlock detection algorithm

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	1	2	1	0
A=	5	1	3	2

Assigned resources

A	0	0	0	0
B	0	1	0	0
C	1	1	1	0
D	0	0	0	0
E	0	0	0	0

Resources still needed

A	-	-	-	-
B	0	1	1	2
C	3	1	0	0
D	-	-	-	-
E	2	1	1	0

An example for the deadlock detection algorithm

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	1	1	1	0
A=	5	2	3	2

Assigned resources

A	0	0	0	0
B	0	0	0	0
C	1	1	1	0
D	0	0	0	0
E	0	0	0	0

Resources still needed

A	-	-	-	-
B	-	-	-	-
C	3	1	0	0
D	-	-	-	-
E	2	1	1	0

An example for the deadlock detection algorithm

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	0	0	0	0
A=	6	3	4	2

Assigned resources

A	0	0	0	0
B	0	0	0	0
C	0	0	0	0
D	0	0	0	0
E	0	0	0	0

Resources still needed

A	-	-	-	-
B	-	-	-	-
C	-	-	-	-
D	-	-	-	-
E	2	1	1	0

An example for the deadlock detection algorithm

Deadlock Detection and Recovery
Banker's Algorithm for Multiple Resources

SAFE

	Tape drivers	Plotters	Scanners	CD-Roms
E=	6	3	4	2
P=	5	3	2	2
A=	1	0	2	0

Assigned resources

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 still needed

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

An example for the deadlock detection algorithm

Deadlock Avoidance
Practical Avoidance

- Two Phase Locking
  - Phase I
    - Process tries to lock all resources it needs, one at a time
    - If needed resources found locked, start over
    - (no real work done in phase one)
  - Phase II
    - Run
    - Releasing locks

- Note similarity to requesting all resources at once
- Algorithm works where programmer can arrange

Deadlock Prevention
R: Conditions for Deadlock

1. Mutual exclusion condition
   - Each resource assigned to 1 process or is available
2. Hold and wait condition
   - Process holding resources can request additional
3. No preemption condition
   - Previously granted resources cannot forcibly taken away
4. Circular wait condition
   - Must be a circular chain of 2 or more processes
   - Each is waiting for resource held by next member of the chain

Deadlock Prevention
Mutual Exclusion Condition

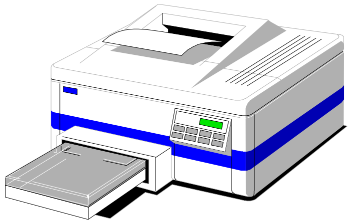
- Some resources are not sharable
  - Printer, tape, etc
- Some resources can be made sharable
- Some resources can be made virtual
  - Spooling - Printer
    - Does spooling apply to all non-sharable resources?
  - Mixing - Soundcard
- Principle:
  - Avoid assigning resource when not absolutely necessary
  - A few processes as possible actually claim the resource

Deadlock Prevention
Hold and Wait Condition

- Require processes to request resources before starting
  - A process never has to wait for what it needs
  - Telephone companies do this
- Problems
  - May not know required resources at start of run
  - Also ties up resources other processes could be using
- Variation:
  - Process must give up all resources
  - Then request all immediately needed

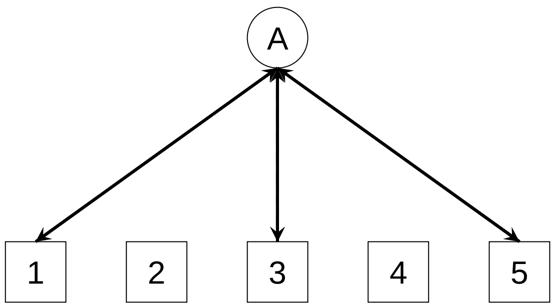
Deadlock Prevention
No Preemption Condition

- This is not a viable option
- Consider a process given the printer
  - Halfway through its job
  - No forcibly take away printer
  - !!??



Deadlock Prevention
Circular Wait Condition

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



- Normally ordered resources
- A resource graph

Deadlock Prevention
Circular Wait Condition

- Impose an order of requests for all resources
- Method
  - Assign a unique id to each resource
  - All resource requests must be in an ascending order of the ids
  - Release resources in a descending order
- Can you prove this method has no circular wait?
- Is this generally feasible?

Deadlock Prevention
Overview

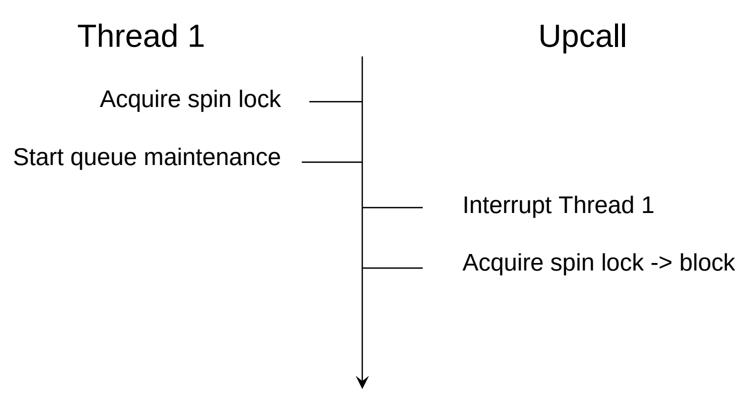
Condition	Approach
Mutual exclusion	Spool everything
Hold and wait	Request all resource initially
No preemption	Take resources away
Circular wait	Order resources numerically

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

Preempting Scheduler Activations

- So how do they handle this deadlock?



Preempting Scheduler Activations

- Detection and recovery (like in Mach)
- Basic idea:
  - Upcall handler checks first the state of each interrupted thread
  - If it is in a critical section allow it to finish this
- Implementation:
  - Protect critical sections with spin locks
  - `Acquire_spin_lock()` increments a counter in the thread's descriptor
  - Upcall handler checks spin lock count of all interrupted threads
  - If there are threads that hold spin locks set flag
  - Switch to the context of these threads
  - Afterwards switch back to upcall handler

Summary

- Resource
- Introduction to deadlocks
- Strategies
  - Ostrich algorithm
  - Deadlock detection and recovery
  - Deadlock avoidance
  - Deadlock prevention
- Non-resource deadlocks