


COS 318: Operating Systems

Deadlock



Jaswinder Pal Singh
Computer Science Department
Princeton University

(<http://www.cs.princeton.edu/courses/cos318/>)



Today's Topics



- ◆ Conditions for deadlock
- ◆ Strategies to deal with deadlocks



2

Definitions



- ◆ Use "processes" and "threads" interchangeably
- ◆ Resource: a (passive) thing needed by thread to do its job
 - Preemptable: CPU, Memory (can be taken away)
 - Non-preemptable: Disk, files, mutex, ... (can't be taken away)
- ◆ Operations on a resource: Request, Use, Release
- ◆ Starvation: At least one thread waits forever for resource
 - ◆ Deadlock: A set of processes have a deadlock if **every** process in the set is waiting for an event that only another process in the set can cause
- ◆ In general, deadlock happens with non-preemptable resources
 - Or resource can be taken away and reallocated to alleviate deadlock



3

Example from CPU Scheduling

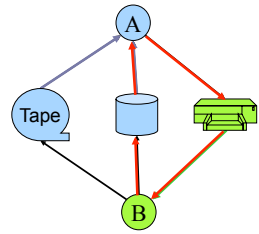
- ◆ T1 at priority 4, T2 at priority 1 and T2 holds lock L
- ◆ T1 needs lock, but for it to get lock it needs T2 to release
- ◆ T2 needs to get on CPU to release lock
- ◆ But T2 does not get CPU until T1 gets lock and makes progress and gives up CPU, and T1 does not get lock until T2 gets CPU



4

Another Example

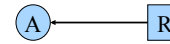
- ◆ A utility program
 - Copy a file from tape to disk
 - Print the file to printer
- ◆ Resources
 - Tape
 - Disk
 - Printer
- ◆ A deadlock
 - A holds tape and disk,
 - B holds printer,
 - A requests for a printer
 - B requests for tape and disk



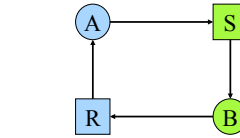
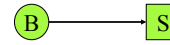
5

Resource Allocation Graph

- ◆ Process A is holding resource R
- ◆ Example: A requests for S while holding R, and B requests for R while holding S, then



- ◆ Process B requests resource S



- ◆ A cycle in resource allocation graph \Rightarrow deadlock

How do you deal with multiple instances of a resource?



6

Conditions for Deadlock

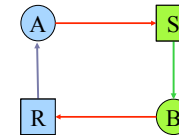
- ◆ Mutual exclusion condition
 - A resource is assigned to no more than one process at a time
- ◆ Hold and Wait
 - Processes holding resources can request new resources
- ◆ No preemption
 - Resources cannot be taken away
- ◆ Circular chain of requests
 - One process waits for another in a circular fashion
- ◆ Question
 - Are all conditions necessary?



7

Eliminate Competition for Resources?

- ◆ If run A to completion and then run B, there will be no deadlock
- ◆ Generalize this idea for all processes?
- ◆ Is this a good idea for CPU scheduling?



Previous example



8

Strategies

- ◆ Ostrich Algorithm
- ◆ Detection and recovery
 - Fix the problem afterwards
- ◆ Dynamic avoidance
 - Careful allocation
- ◆ Prevention
 - Negate one of the four conditions



9

Ignore the Problem

- ◆ The OS kernel locks up
 - Reboot
- ◆ Device driver locks up
 - Remove the device
 - Restart
- ◆ An application hangs (“not responding”)
 - Kill the application and restart
 - Familiar with this?
- ◆ An application runs for a while and then hangs
 - Checkpoint the application
 - Change the environment (reboot OS)
 - Restart from the previous checkpoint



10

Detection and Recovery

- ◆ Detection
 - Scan resource graph
 - Detect cycles
- ◆ Recovery (difficult)
 - Kill process/threads (can you always do this?)
 - Roll back actions of deadlocked threads and retry
 - E.g. transactions: all operations are provisional until they have the required resources to complete operation



11

Deadlock Avoidance

- ◆ Always maintain the Safety Condition:
 - Not currently deadlocked
 - There is some scheduling order in which every process can run to completion (even if all request their max resource needs at once)
- ◆ Banker's algorithm (Dijkstra 65)
 - Single resource
 - Every process has a credit
 - Total resources may not satisfy all credits
 - Track resources assigned to and needed by each process
 - Check on every allocation for safety condition
 - Multiple resources
 - Two matrices: “allocated” and “needed”
 - See textbook for details



12

Examples (Single Resource)

Total: 8

	Has	Max		Has	Max		Has	Max		Has	Max		Has	Max
P ₁	2	6	P ₁	2	6	P ₁	2	6	P ₁	2	6	P ₁	2	6
P ₂	2	3	P ₂	3	3	P ₂	0	0	P ₂	0	0	P ₂	0	0
P ₃	3	5	P ₃	3	5	P ₃	3	5	P ₃	5	5	P ₃	0	0

Free: 1

Free: 0

Free: 3

Free: 1

Free: 6

	Has	Max
P ₁	4	6
P ₂	1	3
P ₃	2	5

?

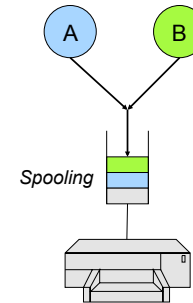
Free: 1



13

Prevention: Avoid Mutual Exclusion

- Some resources are not physically sharable
 - Printer, tape, etc
- Some can be made sharable
 - Read-only files, memory, etc
 - Read/write locks
- Some can be virtualized by spooling
 - Use storage to virtualize a resource into multiple resources
 - Use a queue to schedule
 - Does this apply to all resources?
- What about the tape-disk-printer example?



14

Prevention: Avoid Hold and Wait

- Two-phase locking
 - Phase I:
 - Try to lock all resources at the beginning
 - Phase II:
 - If successful, use the resources and release them
 - Otherwise, release all resources and start over
- What about the tape-disk-printer example?



15

Prevention: No Preemption

- Make the scheduler be aware of resource allocation
- Method
 - If the system cannot satisfy a request from a process holding resources, preempt the process and release all resources
 - Schedule it only if the system satisfies all resources
- Alternative
 - Preempt the process holding the requested resource
- Copying
 - Copying to a buffer to release the resource?
- What about the tape-disk-printer example?



16

Prevention: No Circular Wait

- ◆ Impose an order of requests for all resources
- ◆ Method
 - Assign a unique id to each resource
 - All requests must be in an ascending order of the ids
- ◆ A variation
 - Assign a unique id to each resource
 - No process requests a resource lower than what it is holding
- ◆ What about the tape-disk-printer example?
- ◆ Can we prove that this method has no circular wait?



17

Which Is Your Favorite?

- ◆ 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)
 - Avoid mutual exclusion
 - Avoid hold and wait
 - No preemption
 - No circular wait



18

Tradeoffs and Applications

- ◆ Ignore the problem for applications
 - It is application developers' job to deal with their deadlocks
 - OS provides mechanisms to break applications' deadlocks
- ◆ Kernel should not have any deadlocks
 - Use prevention methods
 - Most popular is to apply no-circular-wait principle everywhere
- ◆ Other application examples
 - Routers for a parallel machine (typically use the no-circular-wait principle)
 - Process control in manufacturing



19

Summary

- ◆ Deadlock conditions
 - Mutual exclusion
 - Hold and wait
 - No preemption
 - Circular chain of requests
- ◆ Strategies to deal with deadlocks
 - Simpler ways are to negate one of the four conditions



20