COS 318: Operating Systems

Deadlock

Today’s Topics

- Conditions for deadlock
- Strategies to deal with deadlocks

Definitions

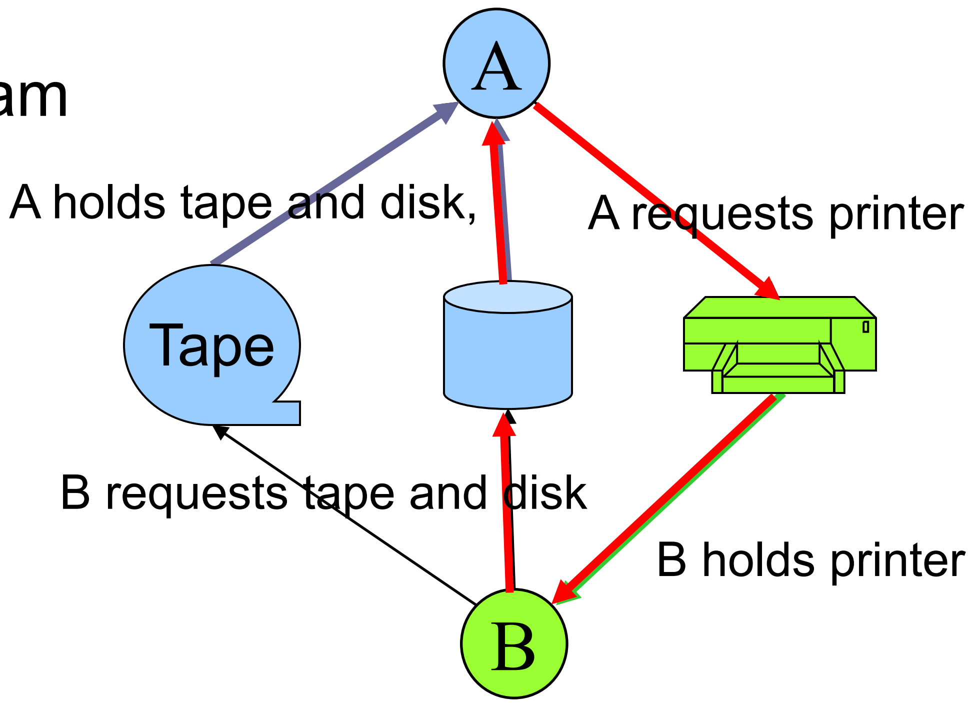
- Use “processes” and “threads” interchangeably (1 thread per proc)
- Resource: a (passive) object that can be granted to a thread and that it needs to do its job
  - Preemptable: CPU, Memory (can be taken away from thread without harm)
  - Non-preemptable: files, mutex, CD recorder ... (can’t just be 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
- Livelock?
- In general, deadlock happens with non-preemptable resources
  - Or resource can be taken away and reallocated to alleviate deadlock

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 T2 must release lock
- 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
- Introducing another thread T3 at priority 3 creates a less contrived situation

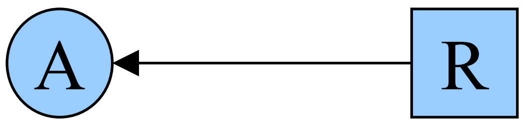
Another Example

- A utility program
  - Copy a file from tape to disk
  - Print the file to printer
  - Two processes running program
- 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

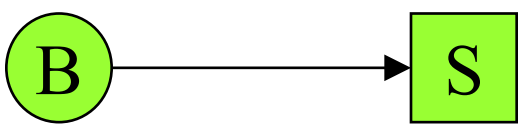


Resource Allocation Graph

- Process A is holding resource R

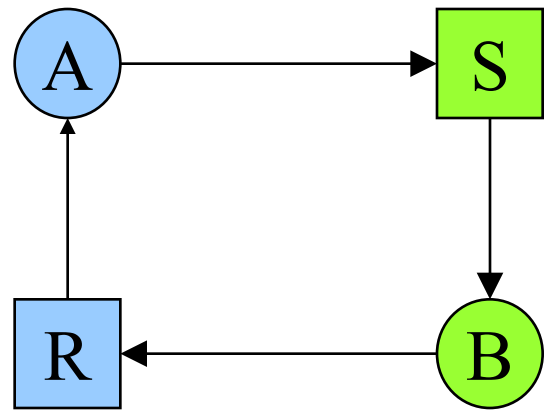


- Process B requests resource S



Example:

- A requests S while holding R



- B requests R while holding S
- A cycle in resource allocation graph ⇒ deadlock

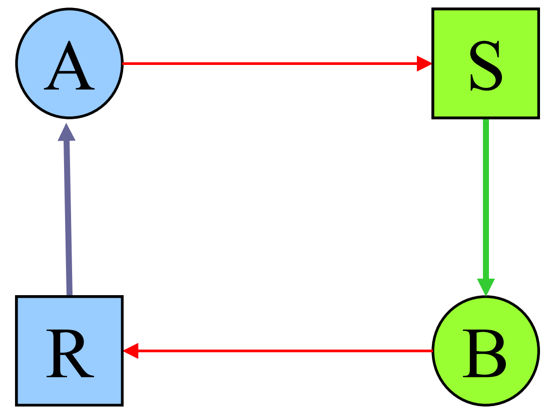
How do you deal with multiple instances of a resource?

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 while continuing to hold the old resources
- No preemption
  - Resources cannot be taken away once obtained
- Circular chain of requests
  - One process waits for another in a circular fashion
- Question
  - Are all conditions necessary?

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

Strategies

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

Ignore the Problem

- The OS kernel locks up
  - Reboot
- Device driver locks up
  - Remove the device
  - Restart
- An application hangs ("not responding")
  - Terminate 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

Detection and Recovery

- Detection
  - Scan resource graph
  - Detect cycles
- Recovery (difficult)
  - Terminate some 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
    - Roll back a process that holds a needed resource to its last checkpoint, releasing resources

Deadlock Avoidance

- Always maintain Safety Condition when allocating resources:
  - 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 type
    - Every process has a credit
    - Total resources may not satisfy all credits
    - Track resources assigned to and needed by each process
    - On every resource allocation, check for Safety Condition

Examples (Single Resource Type)

Total: 8

	Has	Max
P_1	2	6
P_2	2	3
P_3	3	5

Free: 1

	Has	Max
P_1	2	6
P_2	**3**	3
P_3	3	5

Free: **0**

	Has	Max
P_1	2	6
P_2	**0**	**0**
P_3	3	5

Free: **3**

	Has	Max
P_1	2	6
P_2	0	0
P_3	**5**	5

Free: **1**

	Has	Max
P_1	2	6
P_2	0	0
P_3	**0**	**0**

Free: **6**

	Has	Max
P_1	4	6
P_2	1	3
P_3	2	5

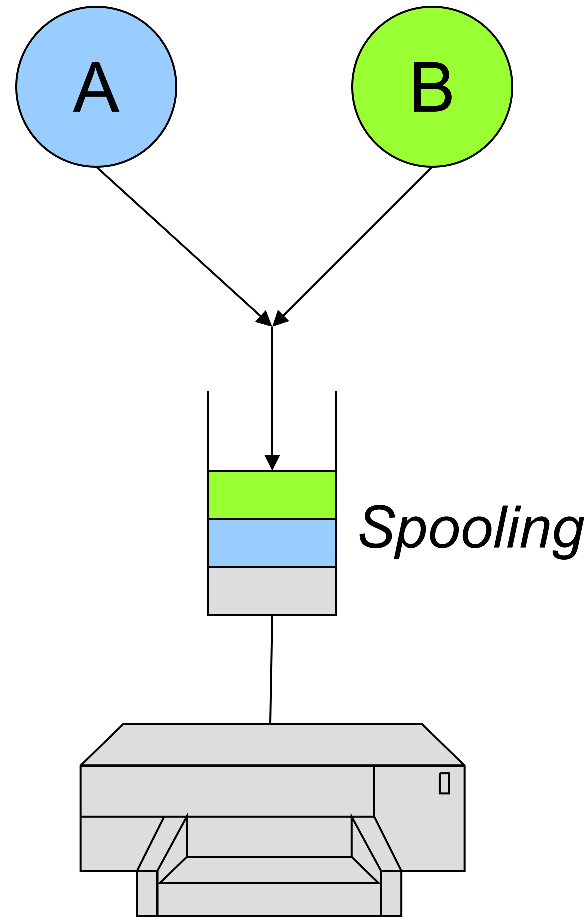
Free: 1

?

- Multiple resource types
  - Two matrices: "allocated" and "needed"
- See textbook for details
- Can we all be bankers and go home?

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, thus eliminating the non-sharable (mutually exclusive) resource from the equation



- What about the tape-disk-printer example?

- Process doesn't have to wait for printer while another process is holding it
  - Move the problem to disk: much bigger

Prevention: Avoid Hold and Wait

- Can't get all resources you need? Don't hold any
- Two-phase locking

  Phase I:
  - Try to lock all resources at the beginning

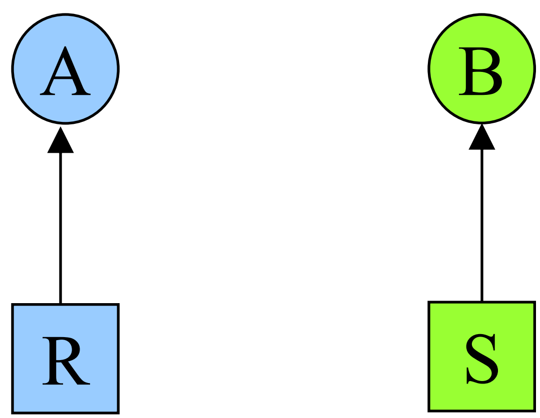
  Phase II:
  - If successful, use the resources and release them
  - Otherwise, release all resources and start over
- What about the tape-disk-printer example?

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?

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?



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

In Practice

- 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

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