Operating Systems

Lecture3.2 - Deadlock Management

Golestan University

Hossein Momeni
momeni@iust.ac.ir

Contents

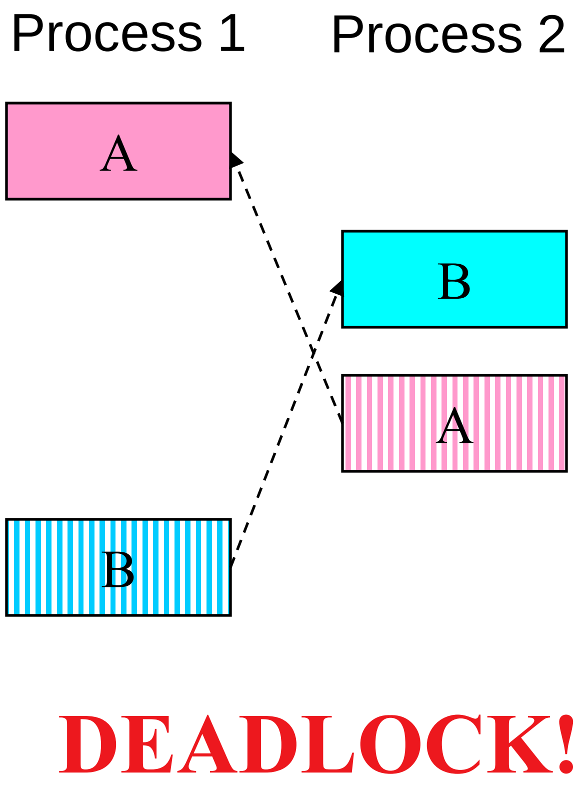
- Resources
- Introduction to deadlocks
  - Why do deadlocks occur?
- Ignoring deadlocks : ostrich algorithm
- Detecting & recovering from deadlock
- Avoiding deadlock
- Preventing deadlock

Resources

- Resource: something a process uses
  - Usually limited (at least somewhat)
- Examples of computer resources
  - Printers
  - Semaphores / locks
  - Tables (in a database)
- Processes need access to resources in reasonable order
- Two types of resources:
  - Preemptable resources: can be taken away from a process with no ill effects
  - Nonpreemptable resources: will cause the process to fail if taken away

When do deadlocks happen?

- Suppose
  - Process 1 holds resource A and requests resource B
  - Process 2 holds B and requests A
  - Both can be blocked, with neither able to proceed
- Deadlocks occur when …
  - Processes are granted exclusive access to devices or software constructs (resources)
  - Each deadlocked process needs a resource held by another deadlocked process



Using resources

- Sequence of events required to use a resource
  - Request the resource
  - Use the resource
  - Release the resource
- Can't use the resource if request is denied
  - Requesting process has options
    - Block and wait for resource
    - Continue (if possible) without it: may be able to use an alternate resource
    - Process fails with error code
  - Some of these may be able to prevent deadlock…

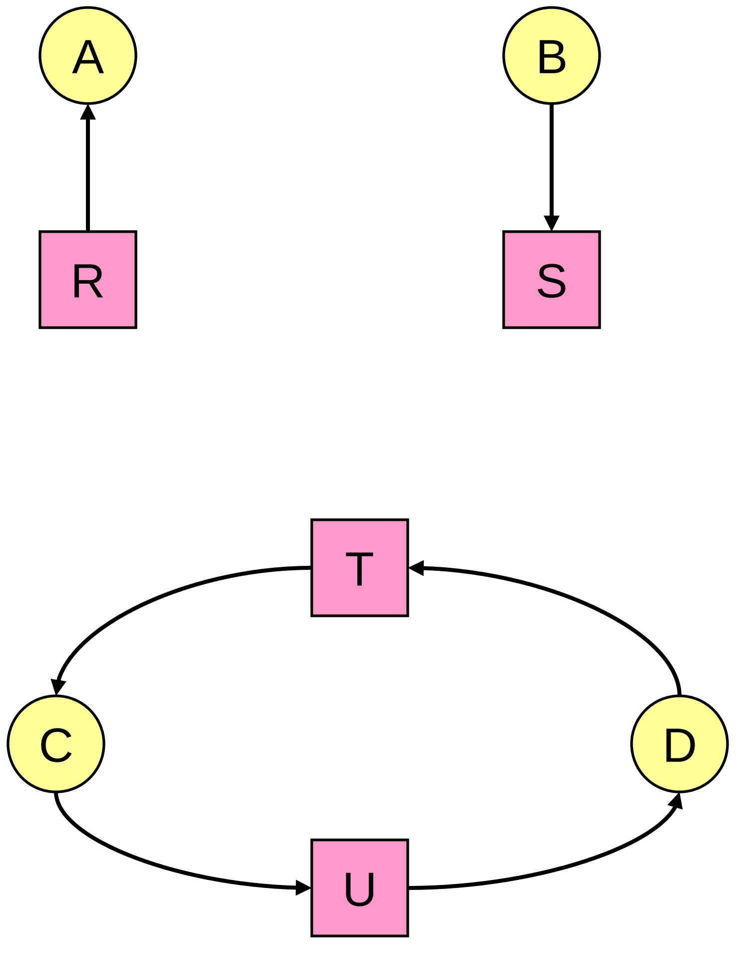
What is a deadlock?

- 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
- In deadlock, none of the processes can
  - Run
  - Release resources
  - Be awakened

Four conditions for deadlock

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

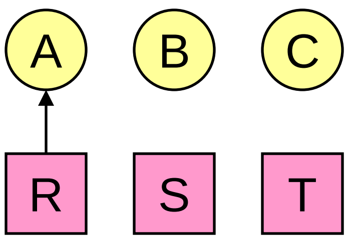
Deadlock Modeling: Resource allocation graphs



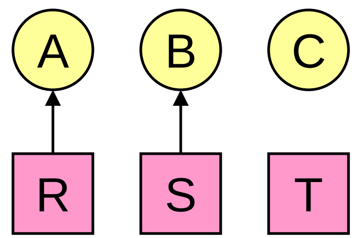
- Resource allocation modeled by directed graphs
- Example 1:
  - Resource R assigned to process A
- Example 2:
  - Process B is requesting / waiting for resource S
- Example 3:
  - Process C holds T, waiting for U
  - Process D holds U, waiting for T
  - C and D are in deadlock!

Getting into deadlock

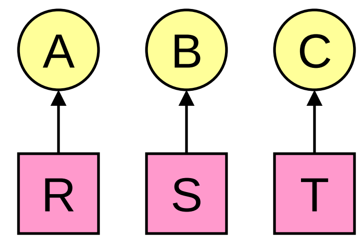
A	B	C
Acquire R	Acquire S	Acquire T
Acquire S	Acquire T	Acquire R
Release R	Release S	Release T
Release S	Release T	Release R



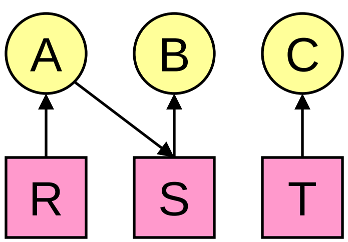
Acquire R



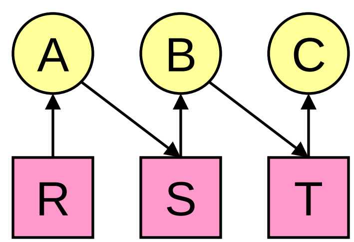
Acquire S



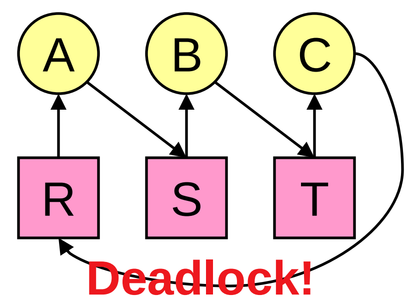
Acquire T



Acquire S

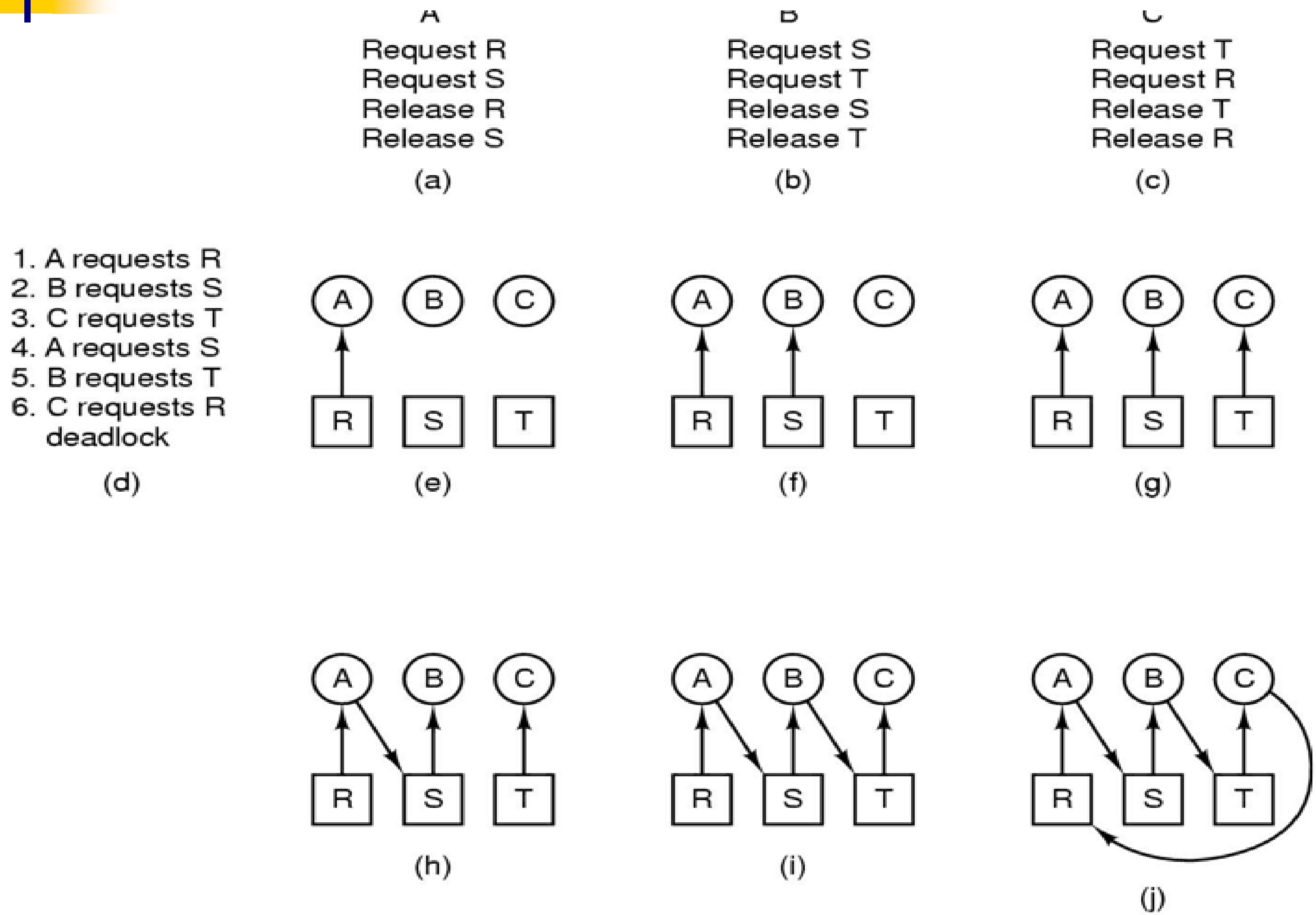


Acquire T



Acquire R

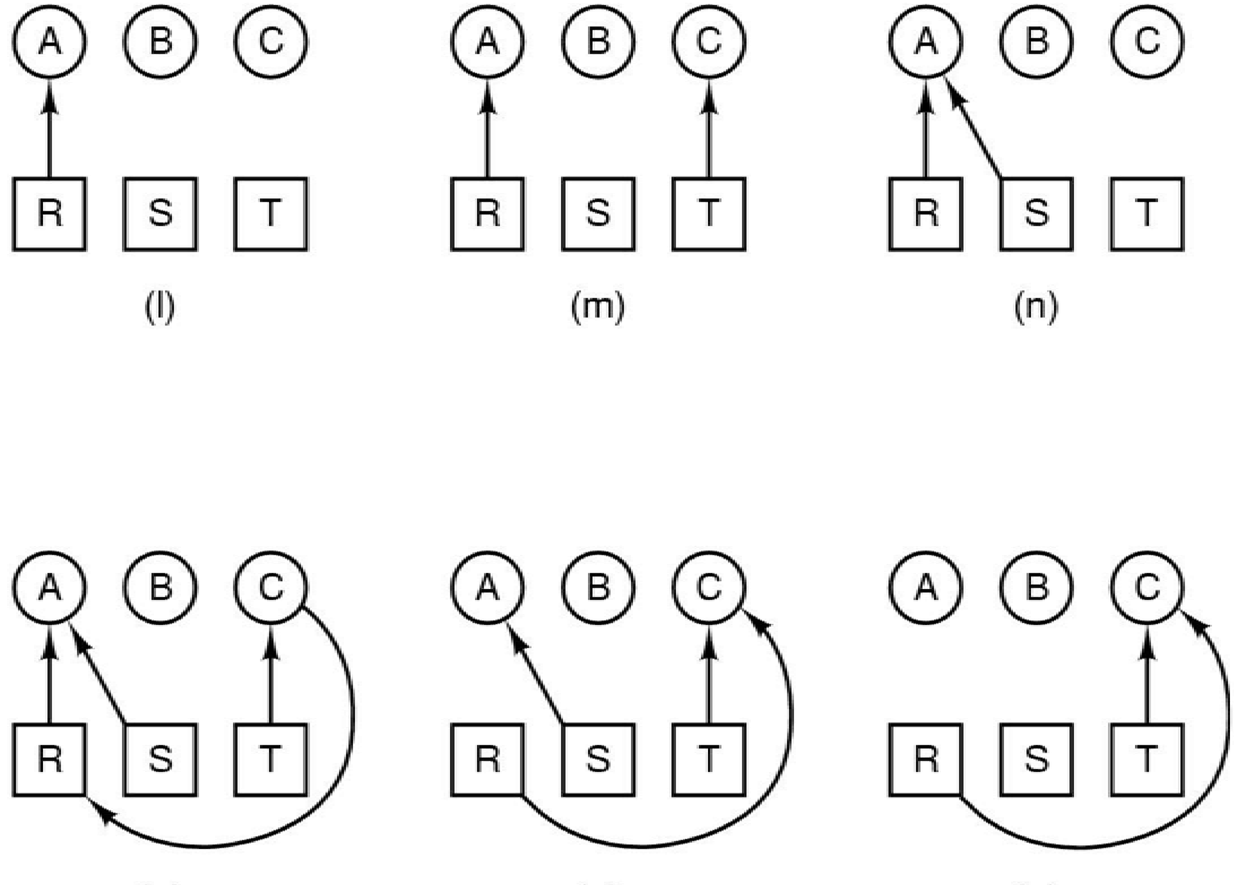
Deadlock Modeling(1)



Deadlock Modeling(2)

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)



Dealing with deadlock

- How can the OS deal with deadlock?
- Four Strategies:
  1. Ignore the problem altogether!
     - Hopefully, it'll never happen…
  2. Detect deadlock & recover from it
  3. Dynamically avoid deadlock
     - Careful resource allocation
  4. Prevent deadlock
     - Remove at least one of the four necessary conditions

- We'll explore these trade offs

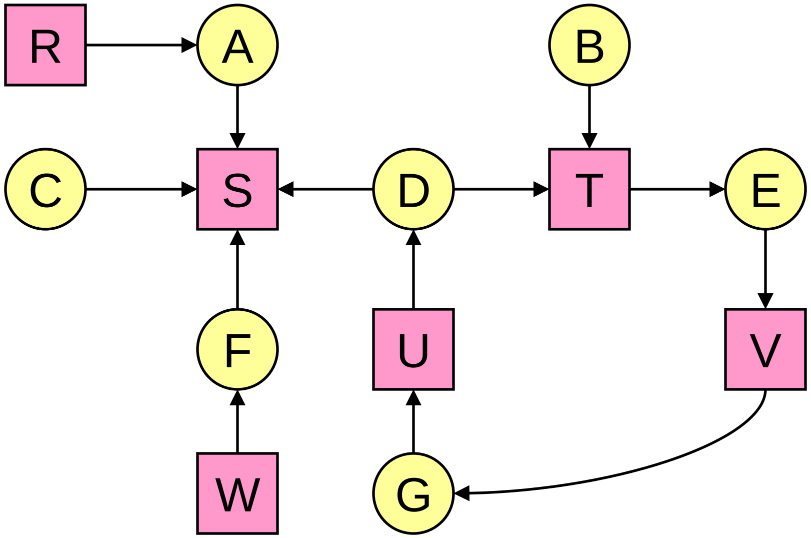
The Ostrich Algorithm

- Pretend there's no problem
- Reasonable if
  - Deadlocks occur very rarely
  - Cost of prevention is high
- UNIX and Windows take this approach
  - Resources (memory, CPU, disk space) are plentiful
  - Deadlocks over such resources rarely occur
  - Deadlocks typically handled by rebooting
- Trade off between convenience and correctness

Detecting deadlocks using graphs

- Process holdings and requests in the table and in the graph (they're equivalent)
- Graph contains a cycle => deadlock!
  - Easy to pick out by looking at it (in this case)
  - Need to mechanically detect deadlock
- Not all processes are deadlocked (A, C, F not in deadlock)

Process	**Holds**	**Wants**
A	R	S
B		T
C		S
D	U	S,T
E	T	V
F	W	S
G	V	U



Deadlock detection algorithm

- General idea: try to find cycles in the resource allocation graph
- Algorithm: depth-first search at each node
  - Mark arcs as they're traversed
  - Build list of visited nodes
  - If node to be added is already on the list, a cycle exists!
- Cycle == deadlock

```
For each node N in the graph {
  Set L = empty list
  unmark all arcs
  Traverse (N,L)
}
If no deadlock reported by now,
there isn't any

define Traverse (C,L)  {
  If C in L, report deadlock!
  Add C to L
  For each unmarked arc from C {
    Mark the arc
    Set A = arc destination
    /* NOTE: L is a
       local variable */
    Traverse (A,L)
  }
}
```

Recovering from deadlock

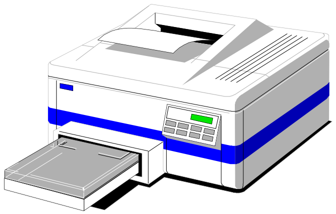
- Recovery through preemption
  - Take a resource from some other process
- Recovery through rollback
  - Checkpoint a process periodically
  - Use this saved state to restart the process if it is found deadlocked
  - May present a problem if the process affects lots of “external” things
- Recovery through killing processes
  - Simplest way to break a deadlock: kill one of the processes in the deadlock cycle
  - Other processes can get its resources
  - Preferably, choose a process that can be rerun from the beginning
    - Pick one that hasn’t run too far already

Deadlock Prevention

- Put suitable restrictions on processes so deadlocks are impossible.
- Four deadlock conditions stated by Coffman, provide a clue for some solutions.
- How to attack these 4 conditions?

Attack to Mutual Exclusion

- If no resource was assigned exclusively to a single process, we would never have deadlocks.
- But how a resource like a printer can be used non-exclusively?
- Answer:
  - By spooling printer output, several processes can generate output at the same time.
  - Only the printer daemon is using the printer.



Attack to Hold and Wait

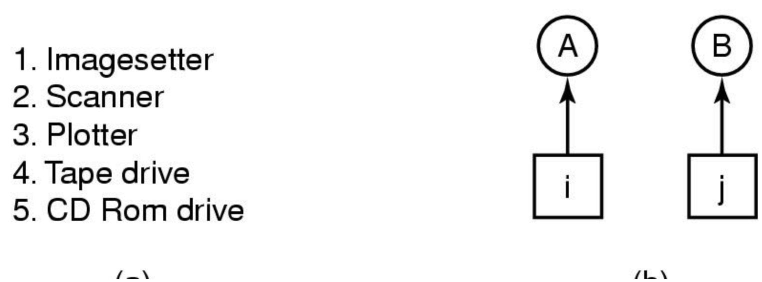
- One way to achieve this goal is to require all processes to request all their resources before starting execution.
- Defects:
  - Many processes do not know which resources they need.
  - Resources will not be used optimally.

Attack to no preemption

- Attacking to this condition is not easy.
- If a process has been assigned the printer and is in the middle of printing, taking away the printer because of a needed plotter is not acceptable.

Attack to circular wait

- A process is allowed to just have one resource at a time.
  - Is it reasonable?
- Another way is to provide a global numbering for resources and all requests must be made in numerical order.



Deadlock Avoidance

- Banker’s Algorithm

- Bankers’ algorithm: before granting a request, ensure that a sequence exists that will allow all processes to complete.
  - Use previous methods to find such a sequence
  - If a sequence exists, allow the requests
  - If there’s no such sequence, deny the request

Banker's Algorithm(Dijkstra,1965) for a single resource

	Has	Max
A	0	6
B	0	5
C	0	4
D	0	7

Free: 10

Any sequence finishes

	Has	Max
A	1	6
B	1	5
C	2	4
D	4	7

Free: 2

C,B,A,D finishes

	Has	Max
A	1	6
B	2	5
C	2	4
D	4	7

Free: 1

Deadlock (unsafe state)

Banker's Algorithm for multiple resources

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

Has

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

Max

E = (6342)
P = (5322)
A = (1020)

E: Existing resources

P: Possessed resources

A: Available resources

Example of banker's algorithm with multiple resources

Example: two-phase locking

- Phase One
  - Process tries to lock all data it needs, one at a time
  - If needed data found locked, start over
  - (no real work done in phase one)
- Phase Two
  - Perform updates
  - Release locks
- Note similarity to requesting all resources at once
- This is often used in databases
- It avoids deadlock by eliminating the “hold-and-wait” deadlock condition