Chapter 3: Deadlock

Overview

- Resources
- Why do deadlocks occur?
- Dealing with deadlocks
    - Preventing deadlock
    - Detecting & recovering from deadlock
    - Avoiding deadlock

Resources

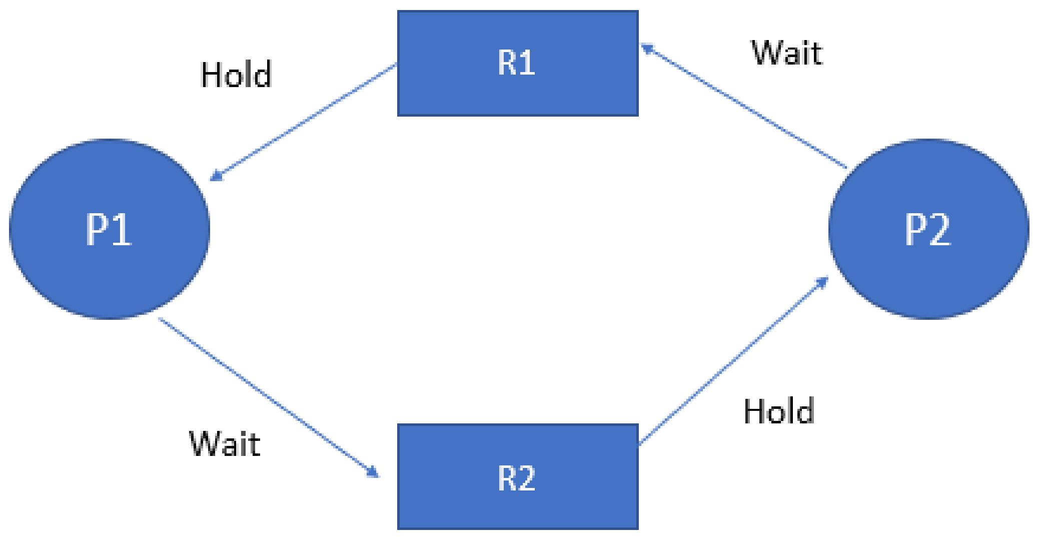
- Resource: something a process uses
  - Usually limited
- Examples of computer resources
  - Printers
  - Files
  - Database
- Processes need access to resources in reasonable order
- Two types of resources:
  - Preemptable resources: can be taken away from a process with no ill effects
  - Nonpreemptable resources: will cause the process to fail if taken away

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

When do deadlocks happen?

- When two or more process acquire a lock in such a way that no process can further progress, that situation is called dead lock



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 is assigned to at most one process
- Hold and wait
  - A process holding resources can request more resources
- Non 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 the next member of the chain

Dealing with deadlock

- How can the OS deal with deadlock?
  - Prevent deadlock
    - Remove at least one of the four necessary conditions
  - Detect deadlock & recover from it
  - Dynamically avoid deadlock
    - Careful resource allocation

Deadlock Prevention

- Deadlock can completely be prevented!
- Ensure that at least one of the conditions for deadlock never occurs
  - Mutual exclusion: Removing mutual exclusion is not a better strategy because it may lost the synchronization.
  - Hold & wait: If all the required resources of a process are allocated at the beginning of the process then no process will wait for new resources in the middle of the execution. Hence, deadlock can be prevented.

    Challenges: (1) Prediction of resources, (2) All required resources may not free/available at the beginning of any process, and (3) Resources utilization falls down.
  - No preemption

- Circular wait:

Step 1: Ordering the resources from 0 to n-1

Step 2: R(i) < R(i+1)

Step 3: If any process holds the lower order resource, it can request the higher order resource. If process holds the higher order resource then its request for lower order resource will be rejected.

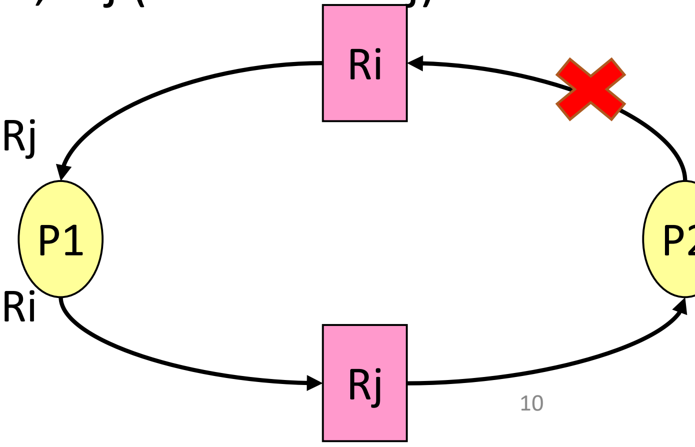
Example: Two processes P1, P2; two resources Ri, Rj (where Ri<Rj)

Event 1# P1 request Ri
consider

Event 2# P2 request Rj
consider

Event 3# P1 request Rj
consider

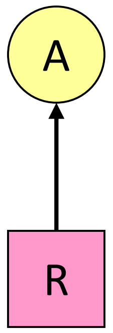
Event 4# P2 request Ri
rejected

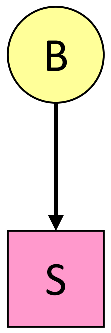


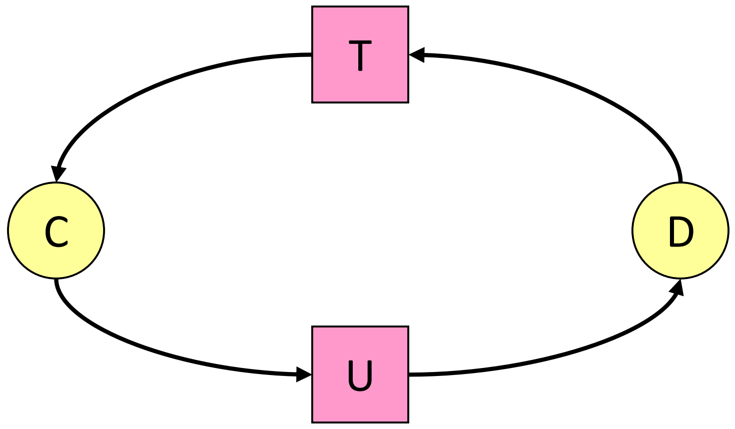
Deadlock Detection & Recovery

- Detect Cyclic/circular wait using resource allocation graph

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

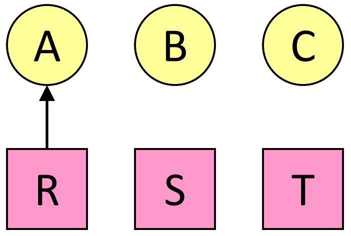
Acquire R
Acquire S
Release R
Release S

B

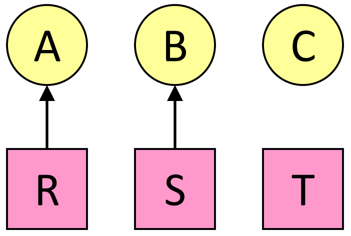
Acquire S
Acquire T
Release S
Release T

C

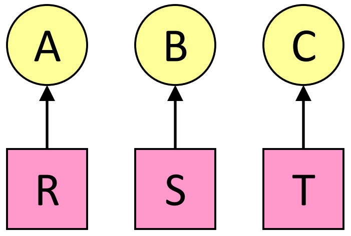
Acquire T
Acquire R
Release T
Release R



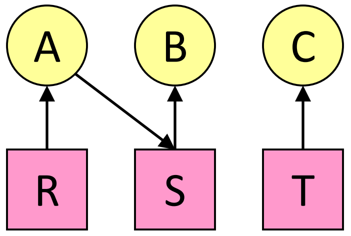
Acquire R



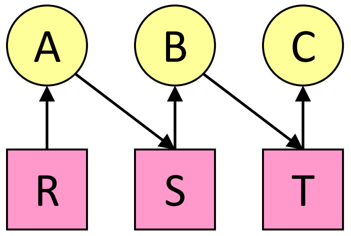
Acquire S



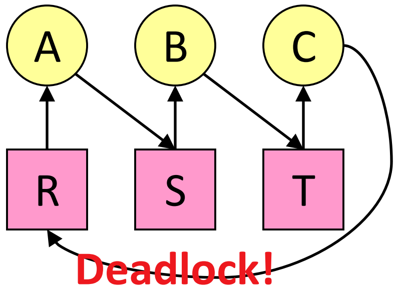
Acquire T



Acquire S



Acquire T

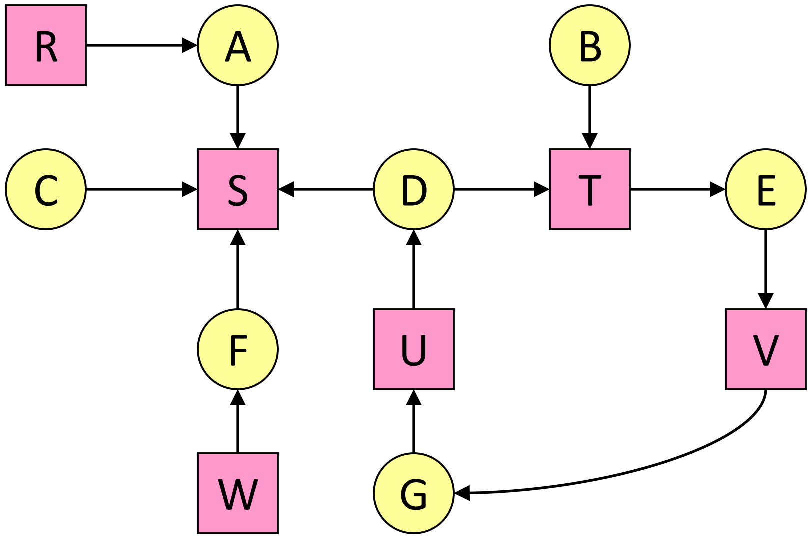


Acquire R

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

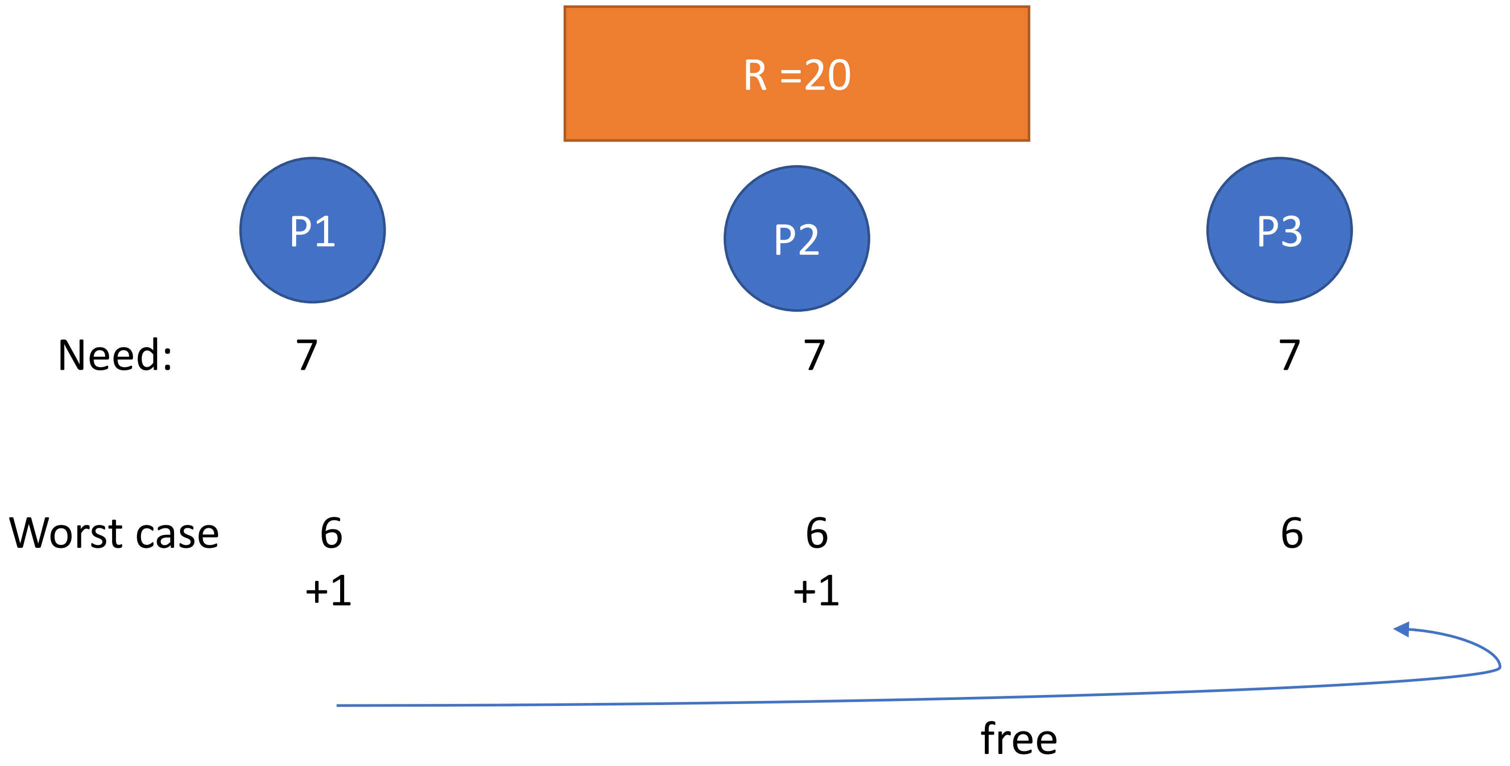
```
For each node N in the graph {
  Set L = empty list
  unmark all arcs
  Traverse (N,L)
}
If no deadlock reported by now,
there isn't any

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)
  }
}
```

Resources with multiple instances

- Previous algorithm only works if there’s one instance of each resource
- If there are multiple instances of each resource, we need a different method
  - Track current usage and requests for each process
  - To detect deadlock, try to find a scenario where all processes can finish
  - If no such scenario exists, we have deadlock

E.g., the system has 20 instances of resource type R. If there are 3 process in the system and each needs 7 instances of R. Check system is in safe state or not.

- Safe state: if the set of process can be driven to completion with the currently available resources, then the system is said to be in safe sate.
- Unsafe state: if above statement is not possible



Ex (2).

No. of processes= 4

Number of instances of R required by each process = 6

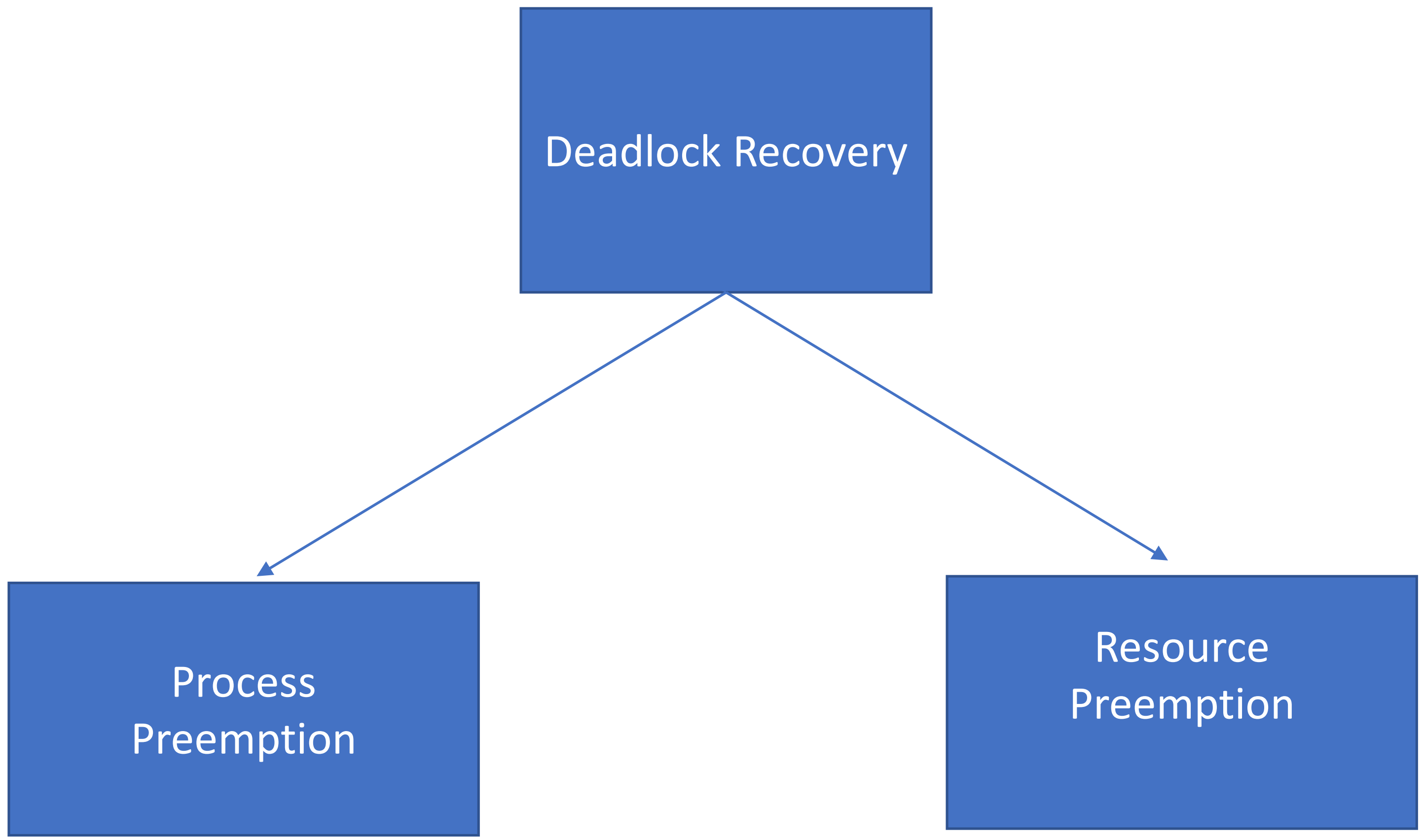
For deadlock free system, minimum R = ??

Ans: 21

Ex (3). A system contains 16 instances of resource R and if each process allowed to request 3 instances. How many process can be processed in deadlock free system.

Ans: 7

Deadlock Recovery



Recovering from deadlock

- Recovery through preemption
  - Take a resource from some other process
  - Depends on nature of the resource and the 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
  - Crudest but 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

To be continued in next lecture

Deadlock Avoidance/Bankers Algorithm

	A	B	C	D
Avail	2	3	0	1

Hold

Process	**A**	**B**	**C**	**D**
1	0	3	0	0
2	1	0	1	1
3	0	2	1	0
4	2	2	3	0

Want

Process	**A**	**B**	**C**	**D**
1	3	2	1	0
2	2	2	0	0
3	3	5	3	1
4	0	4	1	1

Available- A vector of length m indicates the number of available resources of each type.

Allocation- An n*m matrix defines the number of resources of each type currently allocated to a process. Column represents resource and row represents process.

Request- An n*m matrix indicates the current request of each process. If request[i][j] equals k then process P_i is requesting k more instances of resource type R_j.