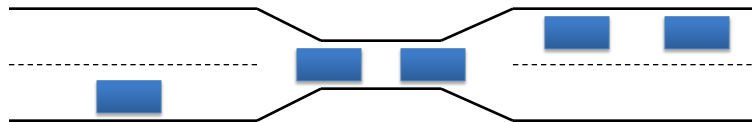# COP 4610
Operating System Principles

Deadlocks

1

Examples of Deadlocks

- Semaphores: mixing up wait & signal
- Kansas (early 20th century): "when two trains approach each other at a crossing, both shall come to a full stop and neither shall start up again until the other has gone"
- System with 2 disks: need both for file transfers
- Single-lane bridge:

2

1

Livelock

- Similar to deadlock, but states of processes change constantly with one another without any of them progressing (special case of resource starvation)

3

System Model

- System consists of resources
- Resource types R_1, R_2, . . ., R_m
  - *CPU cycles, memory space, I/O devices*
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

4

Deadlock Characterization

- **Mutual exclusion:** only one process at a time can use a resource
- **Hold and wait:** a process holding at least one resource is waiting to acquire additional resources held by other processes
- **No preemption:** a resource can be released only voluntarily by the process holding it, after that process has completed its task
- **Circular wait:** there exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, $\ldots$, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

COP 4610 – Operating System Principles   5
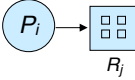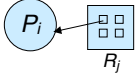
5

Resource-Allocation Graph

- A set of vertices V and a set of edges E.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system

  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system

- **request edge** – directed edge $P_i \rightarrow R_j$

- **assignment edge** – directed edge $R_j \rightarrow P_i$

COP 4610 – Operating System Principles   6
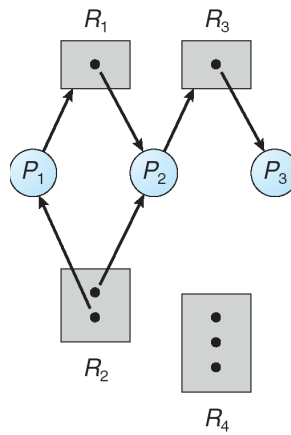
6

Resource-Allocation Graph

- Process

- Resource Type with 4 instances

- P_i requests instance of R_j

  P_i → R_j

- P_i is holding an instance of R_j

  P_i ← R_j

7

Example of a Resource Allocation Graph

R_1 R_3

P_1 P_2 P_3

R_2

R_4

8

4

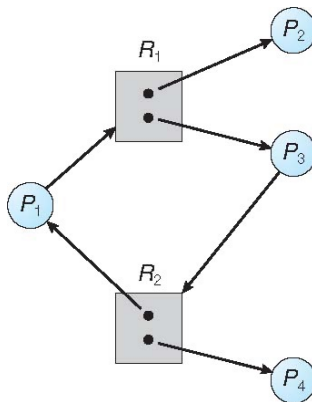# Example of a Resource Allocation Graph
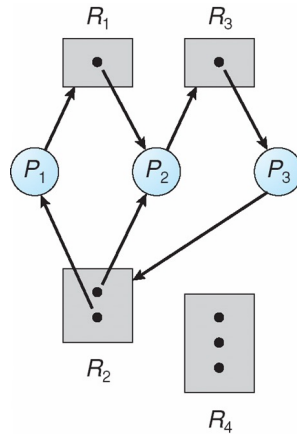
COP 4610 – Operating System Principles 9

9

Another Example

COP 4610 – Operating System Principles 10

10

5

Summary

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

11

Methods for Handling Deadlocks

- Ensure that the system will **never** enter a deadlock state:
  - Deadlock prevention
  - Deadlock avoidence
- Allow the system to enter a deadlock state and then **recover**
- **Ignore** the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX

12

Deadlock Prevention

- Restrain the ways request can be made

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it
  - Low resource utilization; starvation possible

COP 4610 – Operating System Principles   13

13

Deadlock Prevention

- **No Preemption** –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
  - Preempted resources are added to the list of resources for which the process is waiting
  - Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

COP 4610 – Operating System Principles   14

14

Deadlock Avoidance

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state
- System is in **safe state** if there exists a sequence $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_k, with $k < i$
- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_k have finished
  - When P_k is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

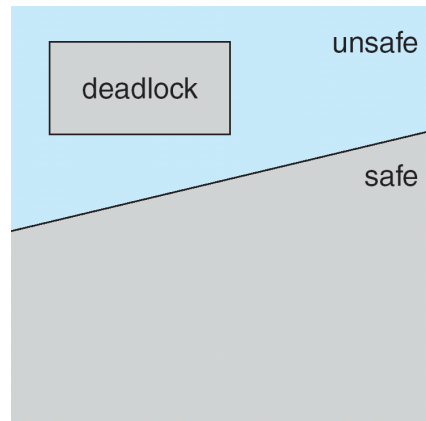COP 4610 – Operating System Principles   15

15

Safe State

- If a system is in safe state $\Rightarrow$ no deadlocks

- If a system is in unsafe state $\Rightarrow$ possibility of deadlock

- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

COP 4610 – Operating System Principles   16

16

Safe State

deadlock

unsafe

safe

17

Avoidance Algorithms

- Single instance of a resource type
  - Use a resource-allocation graph

- Multiple instances of a resource type
  - Use the banker's algorithm

18

Resource-Allocation Graph Scheme

- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_j may request resource R_j; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system

COP 4610 – Operating System Principles   19
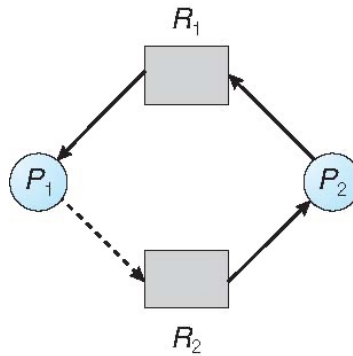
19

Resource-Allocation Graph
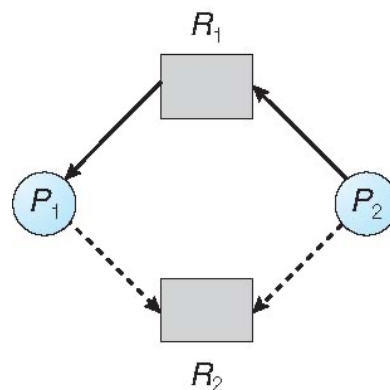
COP 4610 – Operating System Principles   20

20

Resource-Allocation Graph

21

Resource-Allocation Graph Algorithm

- Suppose that process P_i requests a resource R_j
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

22

Deadlock Detection

- Allow system to enter deadlock state

- Detection algorithm

- Recovery scheme

23

Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are processes
  - $P_i \rightarrow P_j$ if P_i is waiting for P_j

- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

- An algorithm to detect a cycle in a graph requires an order of n^2 operations, where n is the number of vertices in the graph

24

Resource-Allocation Graph and Wait-for Graph

(a)

(b)

Resource-Allocation Graph   Corresponding wait-for graph
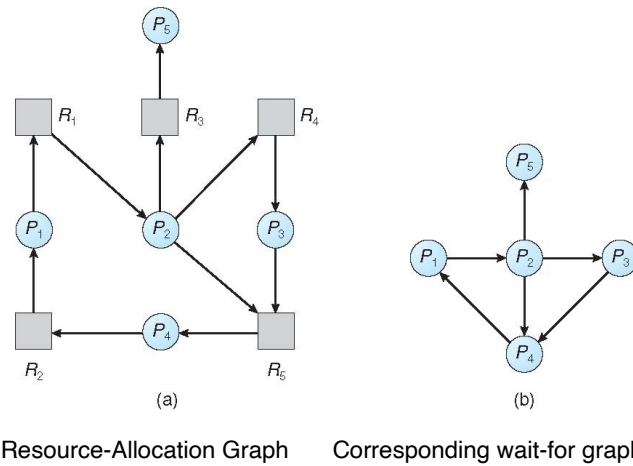
COP 4610 – Operating System Principles   25

25

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes

- Abort one process at a time until the deadlock cycle is eliminated

- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

COP 4610 – Operating System Principles   26

26

Recovery from Deadlock:
Resource Preemption

- **Selecting a victim** – minimize cost

- **Rollback** – return to some safe state, restart process for that state

- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

27