

PROCESSES DEADLOCK (OR DEADLY EMBRACE)

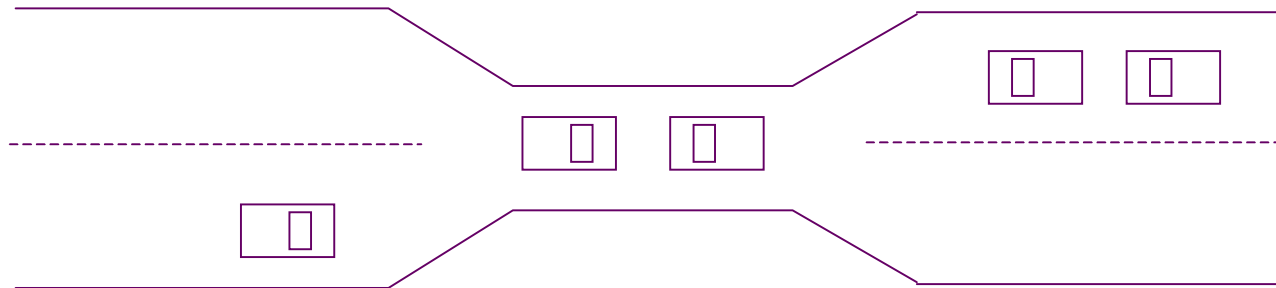
↳ two or more processes are waiting indefinitely for an event that can be caused by only one of the waiting processes.

↳ Tape drives example

- System has 2 tape drives.
- P_1 and P_2 each hold one tape drive and each needs another one.
- semaphores A and B , initialized to 1

P_0	P_1
<i>wait (A);</i>	<i>wait(B)</i>
<i>wait (B);</i>	<i>wait(A)</i>

↳ Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

DEADLOCK SYSTEM MODEL

↪ Resource types R_1, R_2, \dots, 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

↪ Deadlock can arise if four conditions hold simultaneously (**Coffman theorem**)

☞ **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 $\{P_0, P_1, \dots, P_n\}$ 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 , ..., P_{n-1} is waiting for a resource that is held by P_n , and P_0 is waiting for a resource that is held by P_0 .

MODEL REPRESENTATION

Resource-Allocation Graph

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

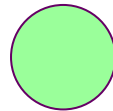
☞ V is partitioned into two types:

- $P = \{P_1, P_2, \dots, P_n\}$, the set consisting of all the processes in the system.
- $R = \{R_1, R_2, \dots, R_m\}$, the set consisting of all resource types in the system.

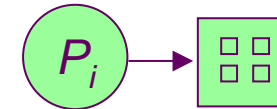
☞ request edge: directed edge $P_i \rightarrow R_j$

☞ assignment edge: directed edge $R_j \rightarrow P_i$

Process



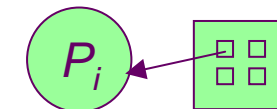
P_i requests instance of R_j



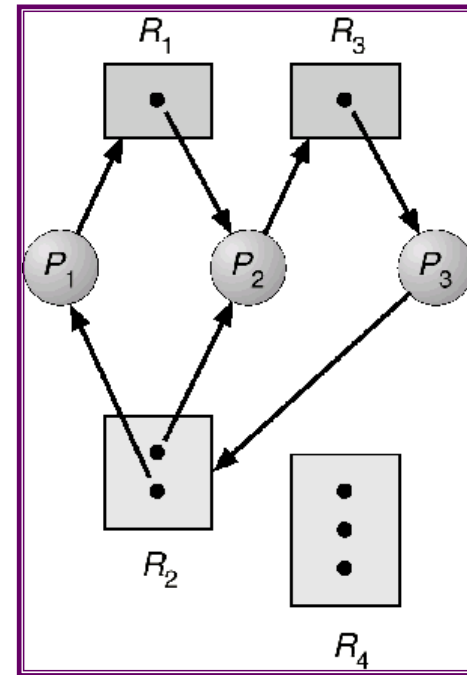
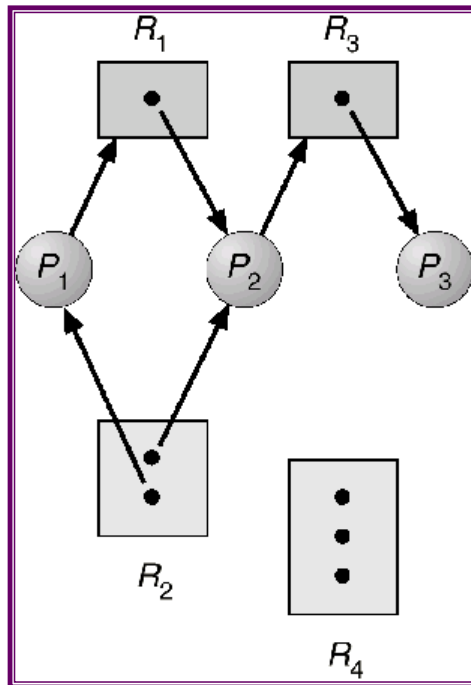
Resource Type with 4 instances



P_i is holding an instance of R_j

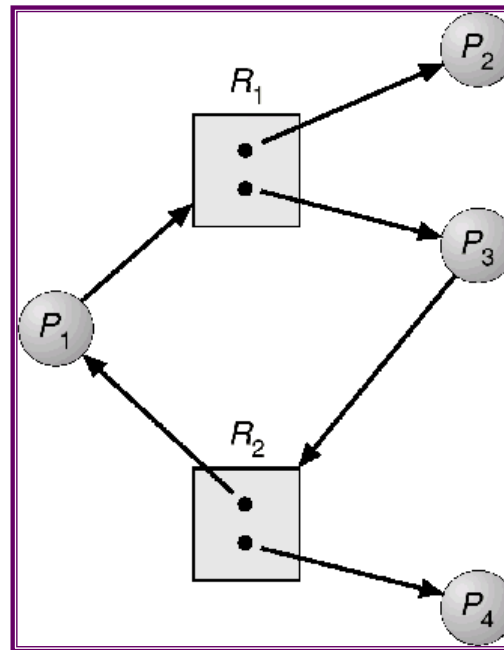


Resource-Allocation Graph examples



Detecting a deadlock in a Resource-Allocation Graph

- ☞ 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.



METHODS FOR HANDLING DEADLOCKS

↳ Ensure that the system will *never* enter a deadlock state

↳ **Prevention**

↳ **Avoidance**

↳ Allow the system to enter a deadlock state and then recover

↳ **Detect and Recover**

↳ Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

DEADLOCK PREVENTION

Resources must be claimed *a priori* in the system (resources pre-allocation)

or

Restrain the ways request can be made (Coffman condition denial).

- ↳ **Mutual Exclusion** – not required for sharable resources; must hold for nonsharable 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.
 - ↳ Low resource utilization; starvation possible.
- ↳ **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.

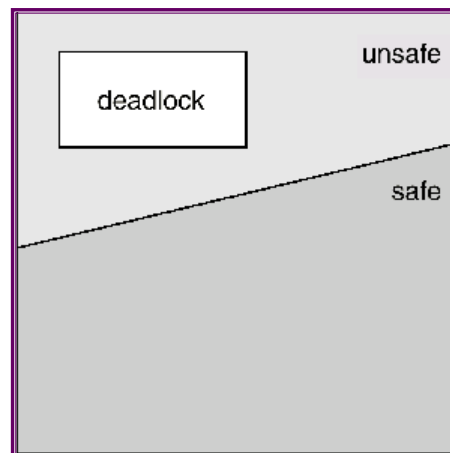
DEADLOCK AVOIDANCE

- ↳ Requires that the system has some additional *a priori* information available.
- ↳ Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.
- ↳ The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.
- ↳ Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes.

SAFE STATE – SAFE SEQUENCE

- ☞ 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 safe sequence of all processes - Banker's Algorithm (Habermann theorem).
- ☞ Sequence $\langle P_1, P_2, \dots, P_n \rangle$ is safe if 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_j , with $j < i$.
 - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished.
 - When P_j 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.
- ☞ 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.



THE BANKER'S ALGORITHM

Example

5 processes P_1 through P_5 ; 3 resource types:
 A (10 instances), B (5 instances), and C (7 instances).

Snapshot at time T_0 :

	<u>Allocation</u>	<u>Max</u>	<u>Available</u>
	A B C	A B C	A B C
P_1	0 1 0	7 5 3	3 3 2
P_2	2 0 0	3 2 2	
P_3	3 0 2	9 0 2	
P_4	2 1 1	2 2 2	
P_5	0 0 2	4 3 3	

	<u>Need</u>
	A B C
P_1	7 4 3
P_2	1 2 2
P_3	6 0 0
P_4	0 1 1
P_5	4 3 1

The system is in a safe state since the sequence $\langle P_2, P_4, P_5, P_3, P_1 \rangle$ satisfies safety criteria.

THE BANKER'S ALGORITHM

Example

P_2 makes a Request₁ = (1,0,2) ≤ Available = (3,3,2) ⇒ true.

	<u>Allocation</u>	<u>Need</u>	<u>Available</u>
	A B C	A B C	A B C
P_1	0 1 0	7 4 3	2 3 0
P_2	3 0 2	0 2 0	
P_3	3 0 2	6 0 0	
P_4	2 1 1	0 1 1	
P_5	0 0 2	4 3 1	

Executing safety algorithm shows that sequence $\langle P_2, P_4, P_5, P_3, P_1 \rangle$ satisfies safety requirement.

Can request for (3,3,0) by P_5 be granted?

Can request for (0,2,0) by P_1 be granted?

DEADLOCK DETECTION

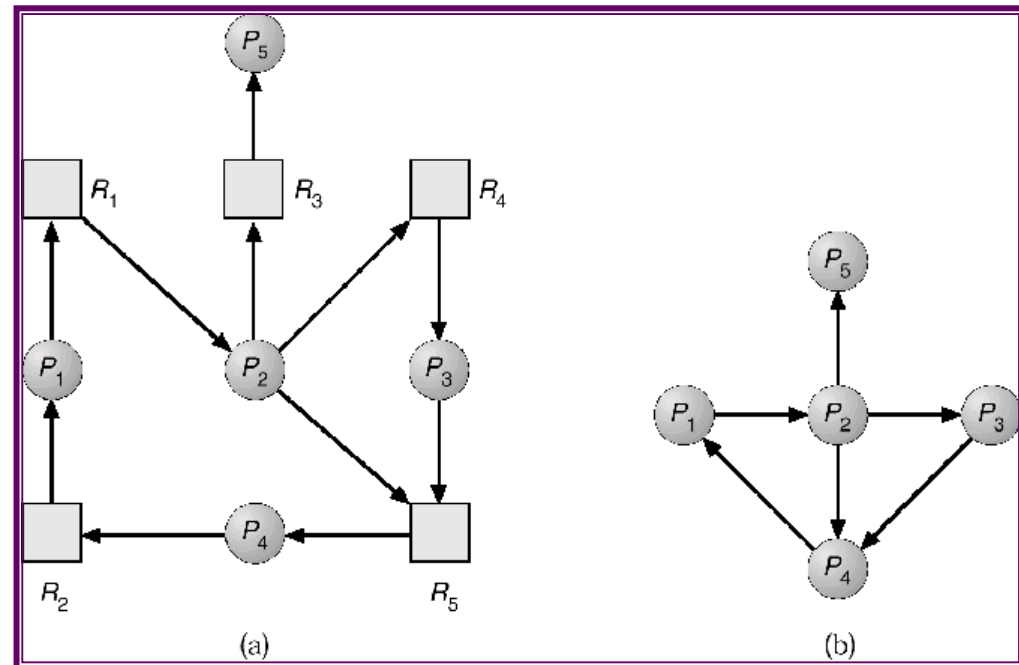
↪ Allow system to enter deadlock state

↪ Detection algorithm

- ❖ 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.
- ❖ 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.

↪ Recovery scheme

- ❖ **Process termination**
- ❖ **Resource Preemption**



DETECTION ALGORITHM

Example

Five processes P_1 through P_5 ; 3 resource types

A (7 instances), B (2 instances), and C (6 instances).

Snapshot at time T_0 :

	<u>Allocation</u>			<u>Request</u>			<u>Available</u>		
	A	B	C	A	B	C	A	B	C
P_1	0	1	0	0	0	0	0	0	0
P_2	2	0	0	2	0	2			
P_3	3	0	3	0	0	0			
P_4	2	1	1	1	0	0			
P_5	0	0	2	0	0	2			

Sequence $\langle P_1, P_3, P_4, P_2, P_5 \rangle$ will result in $Finish[i] = \text{true}$ for all i .

P_3 requests an additional instance of type C.

	<u>Request</u>		
	A	B	C
P_1	0	0	0
P_2	2	0	2
P_3	0	0	1
P_4	1	0	0
P_5	0	0	2

Can reclaim resources held by process P_1 , but insufficient resources to fulfill other processes; requests. Deadlock exists, consisting of processes P_2, P_3, P_4 , and P_5 .

DETECTION-ALGORITHM USAGE

If a system does not use either a deadlock prevention or a deadlock avoidance approach, it

↳ can provide a detection algorithm

↳ must anyway provide a deadlock recovery algorithm.

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?

- ❖ Priority of the process.
- ❖ How long process has computed, and how much longer to completion.
- ❖ Resources the process has used.
- ❖ Resources process needs to complete.
- ❖ How many processes will need to be terminated.
- ❖ Is process interactive or batch?

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.