Operating System

Deadlocks

UNIT-IV

Prepared By

Alok Haldar

Assistant professor

Department of Computer Science & BCA

Kharagpur College

Deadlocks

System model, deadlock characterization, methods for handling deadlocks, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock.

System Model
Deadlock Characterization
Methods for Handling Deadlocks
Deadlock Prevention
Deadlock Avoidance
Deadlock Detection
Recovery from Deadlock
Combined Approach to Deadlock Handling

Deadlocks

A process requests resources; if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called a deadlock.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its designated task. Obviously, the number of resources requested may not exceed the total number of resources available in the system.

For example : A process cannot request three printers if the system has only two.

A process may utilize a resource in only the following sequence:

1. Request : The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
2. Use : The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).
3. Release : The process releases the resource.

The Deadlock Problem

A set of blocked processes each holding a resource and waiting to acquire a resource held by another process in the set.

Example

System has 2 tape drives.

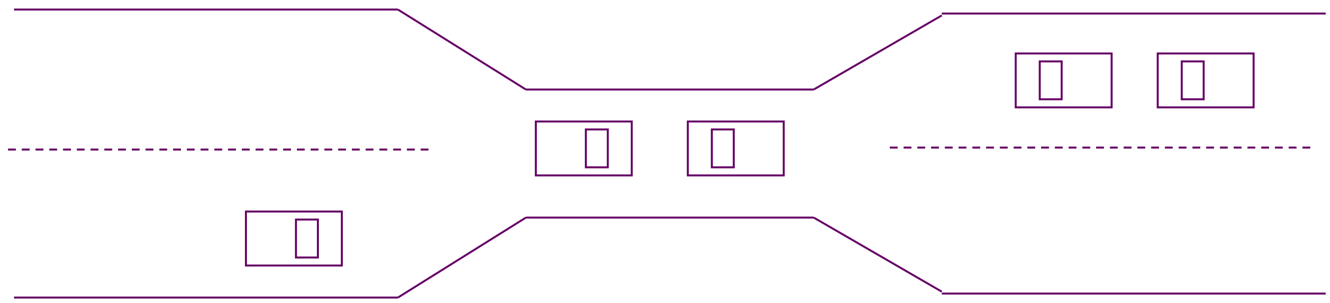
P_1 and P_2 each hold one tape drive and each needs another one.

Example

semaphores *A* and *B*, initialized to 1

P_0	P_1
wait (A);	*wait(B)*
wait (B);	*wait(A)*

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.

System Model

- 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

Deadlock Characterization

Deadlock can arise if four conditions hold simultaneously.

- **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 $\{P_0, P_1, \ldots, P_0\}$ 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

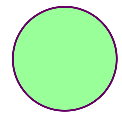
$P_2, \ldots, P_{n-1}$ is waiting for a resource that is held by P_n, and P_0 is waiting for a resource that is held by P_0.

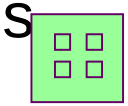
Resource-Allocation Graph

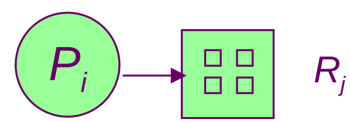
A set of vertices *V* and a set of edges *E*.

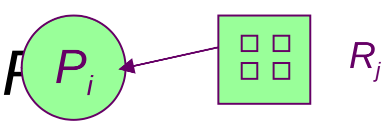
- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system.
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system.
- request edge – directed edge $P_1 \quad R_j$
- assignment edge – directed edge $R_j \quad P_i$

Resource-Allocation Graph (Cont.)

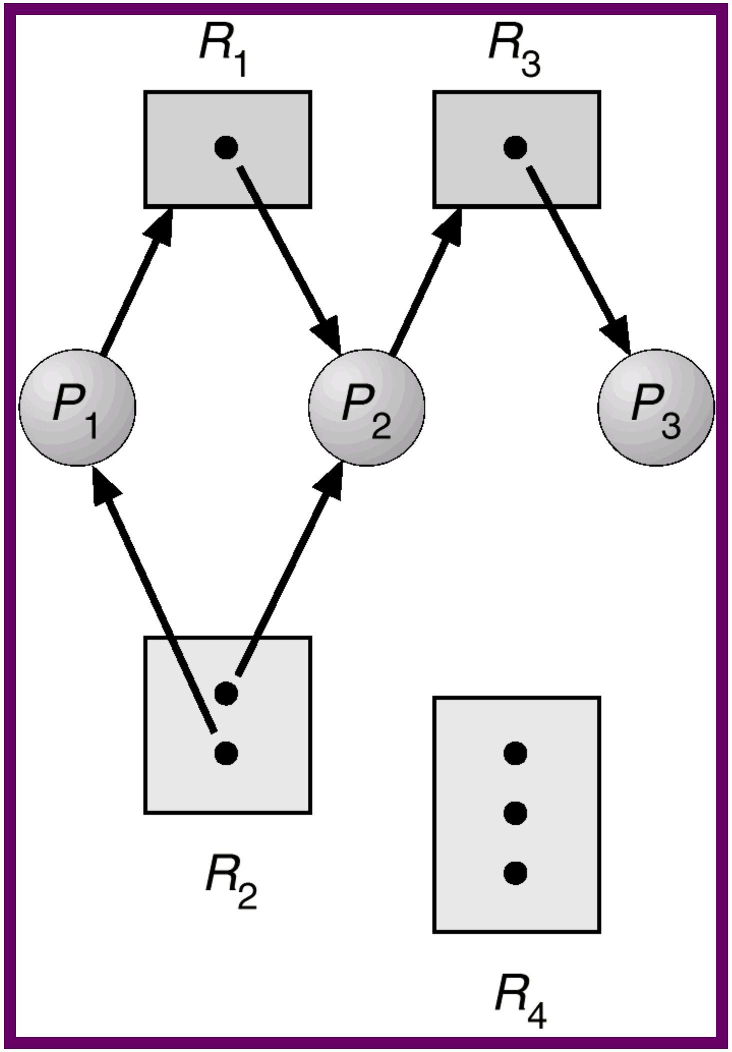
- Process 

- Resource Type with 4 instances 

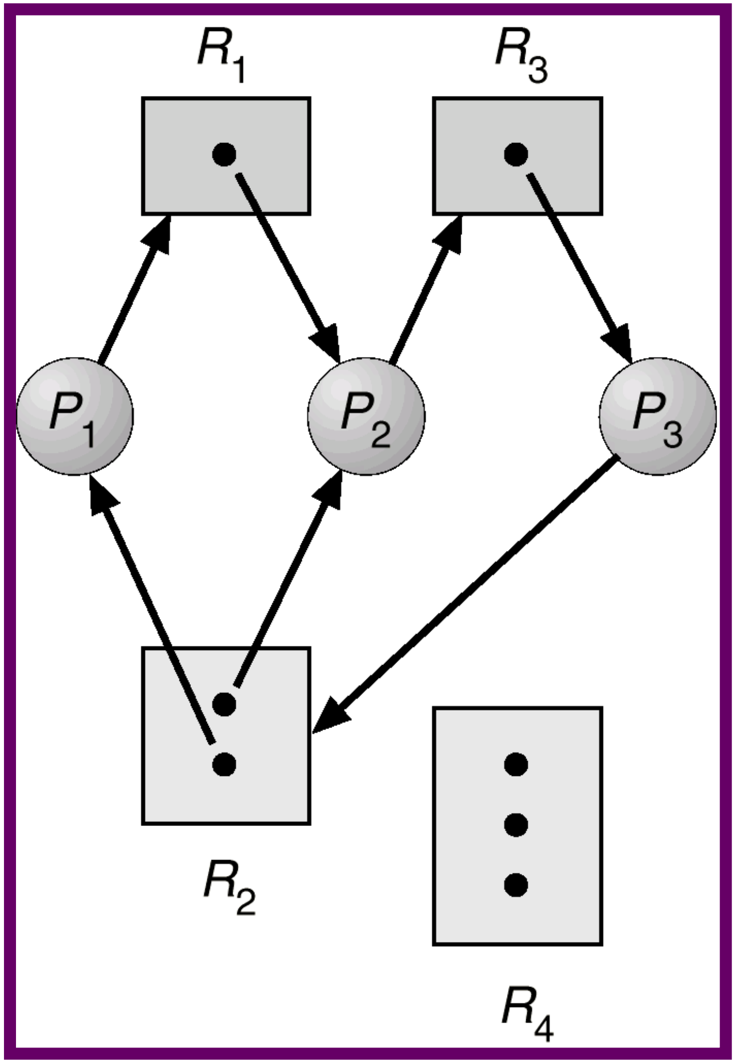
- P_i requests instance of R_j 

- P_i is holding an instance of R_j 

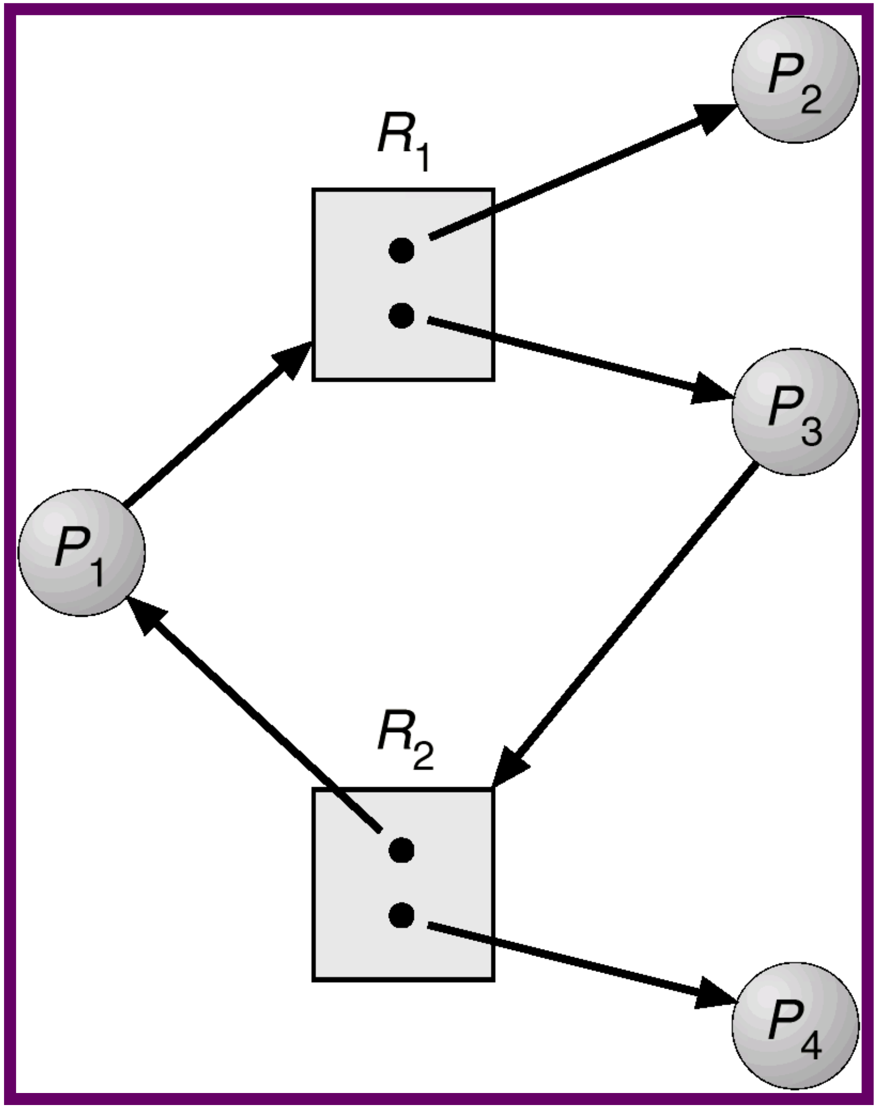
Example of a Resource Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Basic Facts

- If graph contains no cycles no deadlock.

- If graph contains a cycle
  - 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.
- 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.

Deadlock Prevention

How to prevent the deadlock

- **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.

Deadlock Prevention (Cont.)

- **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

- When a process requests an available resource, system must decide if immediate allocation leaves the system in a *safe state*.

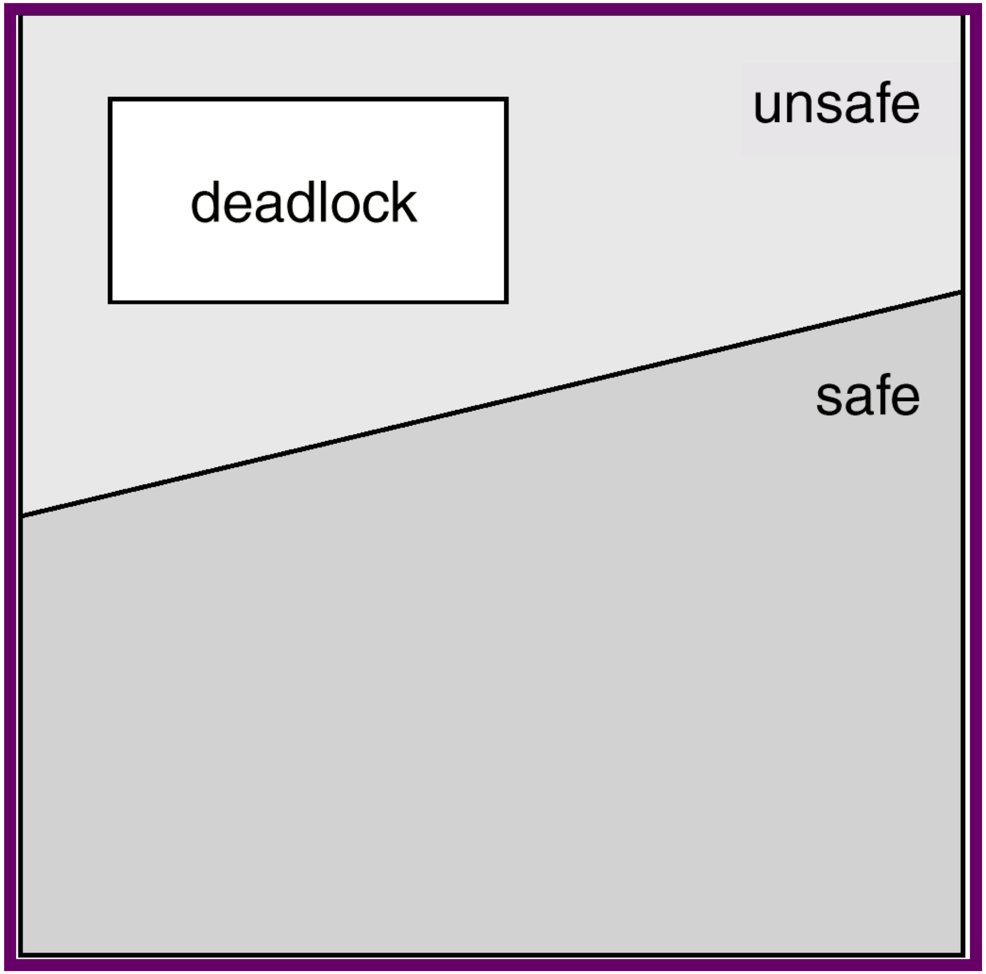
- System is in safe state if there exists a safe sequence of all processes.

- Sequence <P_1, P_2, …, P_n> is safe if for each P_i, the resources that *Pi* 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.

Basic Facts

- If a system is in safe state   no deadlocks.
- If a system is in unsafe state   possibility of deadlock.
- Avoidance   ensure that a system will never enter an unsafe state.

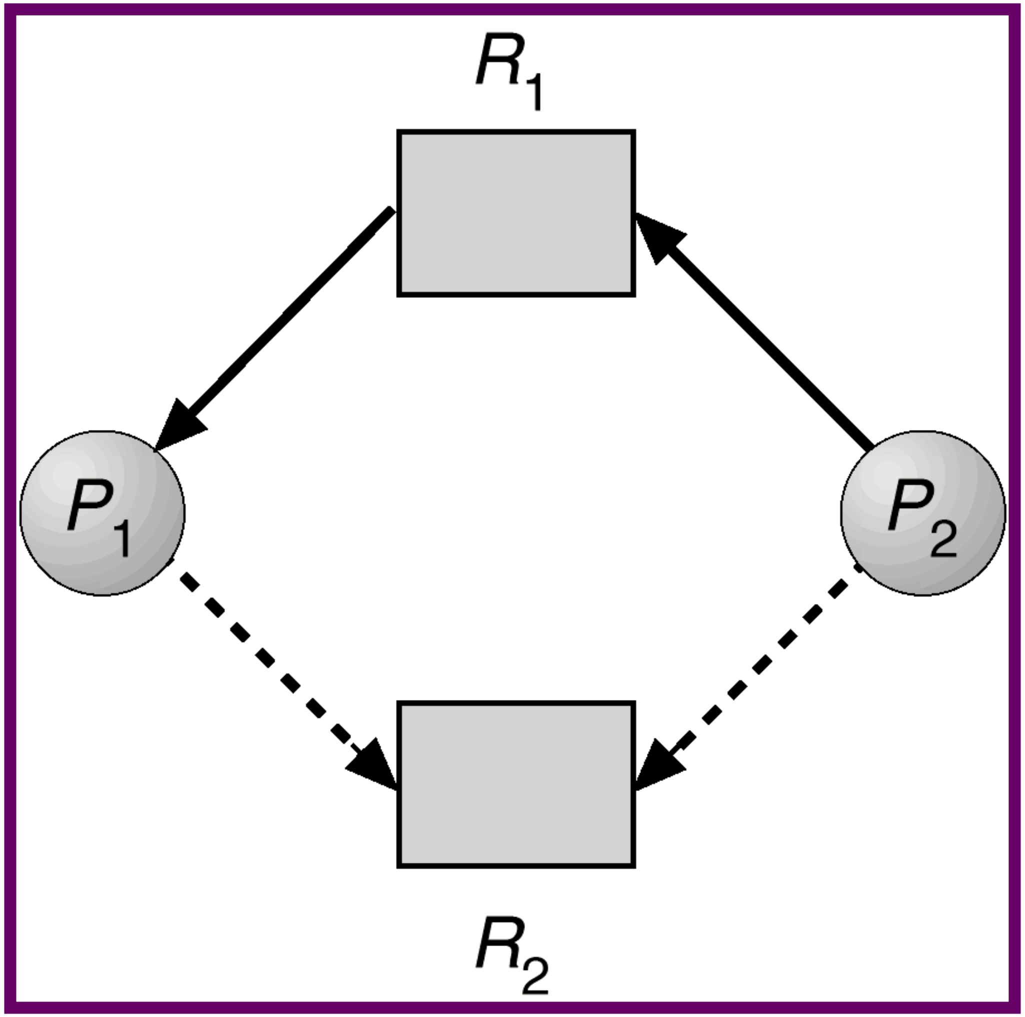
Safe, Unsafe , Deadlock State



Resource-Allocation Graph Algorithm

- *Claim edge* P_i 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.
- When a resource is released by a process, assignment edge reconverts to a claim edge.
- Resources must be claimed *a priori* in the system.

Resource-Allocation Graph For Deadlock Avoidance



Unsafe State In Resource-Allocation Graph