# DEADLOCK

Deadlock is a situation in an operating system where 2 or more processes are blocked forever, each waiting for a resource that is held by another process in the same set. It is a serious problem in operating systems that occurs in a concurrent processing environment where multiple processes complete for limited system resources. Usually occurs in systems where resources such as printers, memory, files and devices are non-sharable and must be used in mutual exclusion mode.

## Necessary Conditions for Deadlock :

1) Mutual Exclusion

It means that at least one resource must be held in a non-sharable mode. Only one process can use the resource at a time.

2) Hold and wait

A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.

3) No Preemption.

Resources cannot be forcibly taken away from a process. A resource can be released only voluntarily by the process holding it after completing its task.

4.) Circular Wait

A set of waiting processes must exist such that each process is waiting for a resource held by the next process in the chain, forming a circular chain of waiting.

## Resource-allocation Graph (RAG)

Deadlocks can be described more precisely in terms of a directed graph called a system resource-allocation graph. The graph consists of a set of vertices V and set of edges E. The set of vertices V is partitioned into 2 different types of nodes $P = \{P_1, P2, ...Pn\}$, the set consisting of all the active processes in the system and $R = \{R1, R2, ..., Rm\}$, the set consisting of all resource types in the system.

The sets P, R and E:

$P = \{P1, P2, P3\}$

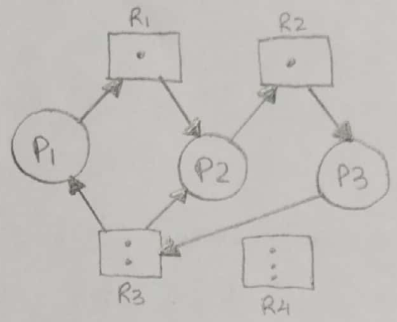
$R = \{R1, R2, R3, R4\}$

$E = \{P1 \rightarrow R1, P2 \rightarrow R3, R1 \rightarrow P2, R2 \rightarrow P2, R2 \rightarrow P1, R3 \rightarrow P3\}$

Resource instances:

One instance of resource type R1
Two instances of resource type R2
One instance of resource type R3
Three instances of resource type R4

Scanned with OKEN Scanner



## Methods for Handling deadlock

1.) Deadlock Prevention or Avoidence
2.) Deadlock Detection and Recovery
3.) Ignoring deadlocks

## Deadlock Prevention

Deadlock Prevention works by ensuring that at least one of the four necessary conditions for deadlock never holds.

1.) Mutual Exclusion cannot always prevented ~~by forcing~~ because some resources are inherently non-sharable.

2.) Hold and wait can be prevented by forcing processes to request all required ~~all~~ resources before execution or allowing resource requests only when the process holds no other resources.

3.) No Preemption can be prevented by preempting resources from waiting processes and allocating them to requesting processes.

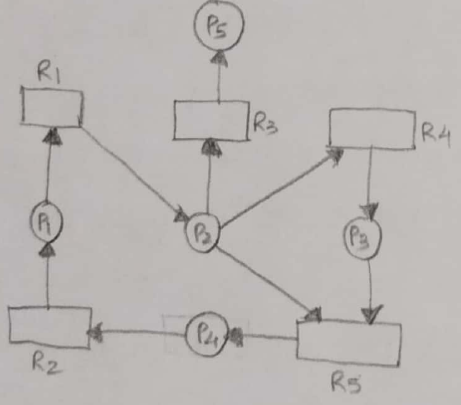
4.) Circular wait can be prevented by imposing a total ordering of resources and forcing processes to request resources in increasing order

## Deadlock Detection

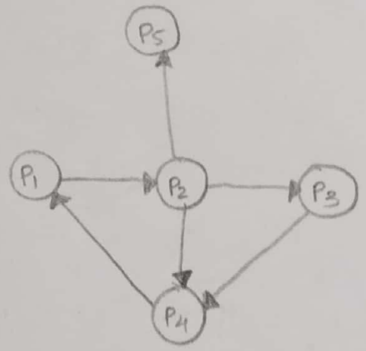
If a system does not employ either a deadlock-prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system must provide. An algorithm that examines the state of the system to determine whether a deadlock has occured.
An algorithm to recover from the deadlock.

### Single instance of each rosource type :-

If all resources have only a single instance then we can define a deadlock detection algorithm that uses a variant of the Resource-allocation graph called a wait-for graph. We obtain this graph from the RAG by removing nodes of type resource and collapsing the appropriate edges.



Resource Allocation Graph



Wait-for graph

## Deadlock Recovery

Once a deadlock is detected, the system can recover using the following methods:

1.) Process Termination

- Abort all deadlocked processes - breaks the deadlock but wastes computation work.
- Abort one process at a time - reduces loss but requires repeated deadlock detection.

2.) Resource Preemption.

Resources are taken from some process and given to others to break the deadlock. This involves :

- Selecting a victim : choosing the process to preemption based on cost factors.
- Rollback : rolling back the process to a safe state and restarting it.
- Starvation control : ensuring the same process is not always chosen as the victim.

## Deadlock Avoidance

The method for avoiding deadlocks is to require additional information about how resources are to be requested. Unlike deadlock prevention, which restricts resource requests deadlock avoidance allows resource allocation only if it keeps the system in a safe state. For this purpose, the operating system requires advance information about the maximum resource requirements of each process. The resource-allocation state is defined by the no. of available and allocated resources and maximum demands of the processes.

Scanned with OKEN Scanner

## Safe State

A system is said to be in a safe state if there exists at least one safe sequence of processes such that each process can obtain its maximum required resources and complete execution without causing a deadlock.

- A safe state is not a deadlock state
- A deadlock state is always unsafe.
- Not all unsafe states are deadlock states, but they may eventually lead to deadlock.

The operating system grants a resource request only if the resulting state is safe. If granting the request leads to an unsafe state, the process is forced to wait.

## Resource Allocation Graph Algorithm

It is a deadlock avoidance method used when there is only one instance of each resource type. It uses a graph with process nodes, resource nodes, and claim edges to represent possible future resource requests. A resource request is granted only if allocating it does not create a cycle in the graph. If a cycle is formed, the request is denied to keep the system in a safe state and avoid deadlock.

## Banker's Algorithm

The resource-allocation graph algorithm is not suitable for systems with multiple instances of each resource type. In such cases Banker's Algorithm is used. It works similarly to a banking system:

→ Each process must declare maximum number of resources it may need.

→ The operating system allocates resources only if it can satisfy the maximum needs of all processes.

The important data structures used:

1.) Available : A vector of length m indicates the no. of available resources of each type. If Available[j]=k there are k instances of resource type Rj available.

2.) Max : An n, x m matrix defines the maximum demand of each process. If Max[i,j]=k, then process Pi may request at most k instance of resource type Ri

3.) Allocation : An n x m matrix defines the no. of resources of each type currently allocated to each process. If Allocation[i,j]=k, then process Pi is currenly allocated k instances of resource type Rj.

4.) Need : An n x m matrix defines the no: of remaining resources need of each process. If Need[i,j]=k, then process Pi may need k more instances of resource type Ri to complete its task. Note that

Need[i,j] = Max[i,j] - Allocation[i,j]

## Safety Algorithm

1.) let work and finish be vectors of length m & n respectively.

Initialize Work := Available and

Finish[i] := false for i = 1, 2, .. n

2.) Find an i such that both

a.) Finish [i] = false

b.) Need i <= work

If no such i exists, go to step 4.

3.) Work := Work + Allocation

Finish [i] := true go to step 2

4.) If Finish[i] = true for all i.

then the system is in a safe state. This algorithm may require an order of m x n2 operations to decide whether a state is safe.