Deadlock:- A deadlock is a situation where a group of processes is permanently blocked as a result of each process having acquired a set of resources needed for its completion and having to wait for release of the remaining resources held by other, thus making it impossible for any of the deadlocked processes to proceed. For example, take system with one tape drive and one plotter. Process P1 request tape drive and process P2 requests the plotter. Both requests are granted. Now P1 requests the plotter (without giving up the tape drive) and P2 request the tape drive (without giving up the plotter). Neither request can be granted so both processes enter deadlock situation.

Necessary Condition of Deadlock: -

Following are the necessary condition for deadlock:

1. Mutual Exclusion: - The shared resources are acquired and used in a mutually exclusive manner, that is by at most one process at a time.
2. Hold and wait: - Each process continues to hold resources already allocated to it while waiting to acquire other resources.
3. No preemption: - Resources granted to a process can be released back to the system only as a result of the voluntary action of that process, the system cannot forcefully revoke them.
4. Circular waiting: - Deadlock process are involved in a circular chain such that each process holds one or more resources being requested by the next process in the chain.

    The simultaneous existence of these conditions defines the state of deadlock. In other words, all four conditions must be present for a deadlock occur. Thus, by requiring all processes to request and acquire their resource in a strictly increasing order of the specified system resources classes.

Deadlock Avoidance: - The basic idea of deadlock avoidance is to grant only those requests for available resources that can not possibly result in a state of deadlock. This strategy is usually implemented by having the resources allocator examine the effects of granting a particular. If granting of the resource can not possibly lead to deadlock, the resource is granted to the requestor. Otherwise the requesting process is suspended until such time when its pending request can be safely granted. This is usually after one or more resources held by other active processes are released.

In order to evaluate the safety of the individual system states, deadlock avoidance requires all process to state [proclaim] their maximum resource requirements prior to execution.

The resource allocator keeps track of the number of allocated and the number of available resources of each type in addition to recording the remaining number of resources pre-claimed but not yet requested by each process.

Deadlock Detection and Recovery:-

Deadlock detection I sonly a part of the deadlock-handling task. Detecting a deadlock only reveals the existence of the problem, the system must then break the deadlock and to ensure that the affected processes can eventually be completed. The first step in deadlock recovery is to identify the deadlocked processes. This gives an edge to detection algorithms that provide an indentation of deadlocked process. The algorithm operates as follows:

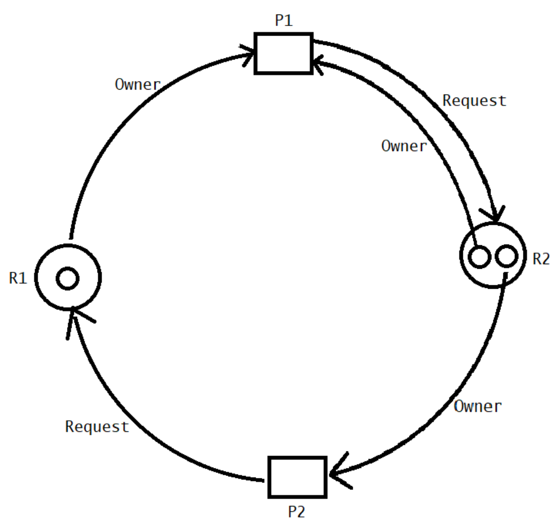
1. Form ALLOCATED, REQUESTED and ALLOCATED in accordance with the system state. Unmark all active processes.
2. Find an unmarked process 'i' such that

   $\text{REQUESTED}^{i} <= \text{AVAILABLE}$

   If found, mark process i, update AVAILABLE

   AVAILABLE:=AVAILABLE + ALLOCATED

   and repeat this step. When no qualifying process can be found. Proceed to next step.
3. If all process is marked, the system is not deadlocked. Otherwise the system is deadlocked.



	R1	R2
P1	1	1
P2	0	1

Allocated

	R1	R2
P1	0	1
P2	1	0

Requested

R1	R2
0	0

Available

The next step is to break the deadlock by rolling back or restarting one or more of the deadlocked processes. Restarting a process implies the loss of the work completed by the process prior to it becoming deadlocked. Since presumably not all processes have progressed equally for, it is desirable to choose the victims among processes whose restarting is les costly. Rolling back a process requires a facility for recording the runtime states of processes, so that a process can be returned to a point sufficiently deep in the past that the deadlock is broken.

Deadlock detection and recovery provides a higher potential degree of concurrency than deadlock prevention and avoidance.