Deadlock

Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
Ex : When two trains are coming towards each other on the same track, none of the trains can move once they are in front of each other.

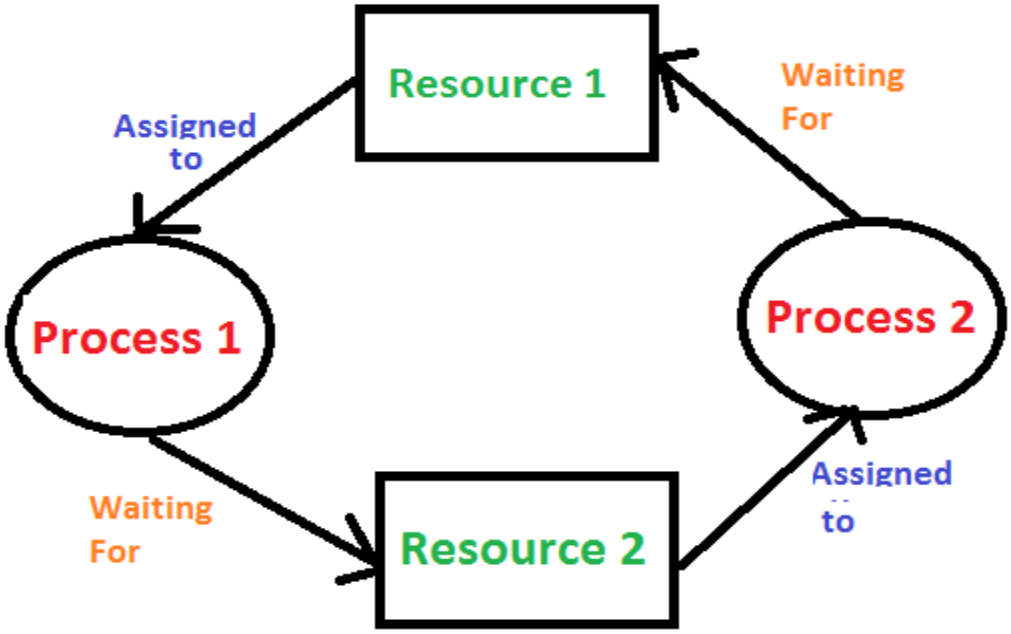


Fig 1

Necessary Conditions for a deadlock to occur:
Mutual Exclusion: Only one process can use at a time.
Hold and Wait: A process is holding at least one resource and waiting for resources.
No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

There are four ways to handle deadlock
1) Deadlock ignorance: Simply ignore the condition. We just assume that there is no deadlock and work as usual. If deadlock is very rare, then let it happen and reboot the system.
2) Deadlock prevention: The idea is to prevent the necessary conditions that are required for a deadlock to occur.

2.1. Eliminate Mutual Exclusion

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

2.2. Eliminate Hold and wait

Allocate all required resources to the process before the start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.

2.3. Eliminate No Preemption

Preempt resources from the process when resources required by other high priority processes.

2.4. Eliminate Circular Wait

Each resource will be assigned with a numerical number. A process can request the resources increasing/decreasing. order of numbering.

3) Deadlock avoidance: At every step, just check whether this resource allocation will either lead to a deadlock or not. Deadlock avoidance can be done with **Banker's Algorithm.**
Banker's Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources, it checks for the safe state, if after granting request system remains in the safe state it allows the request and if there is no safe state it doesn't allow the request made by the process.

Inputs to Banker's Algorithm:

1. Max need of resources by each process.
2. Currently, allocated resources by each process.
3. Max free available resources in the system.

The request will only be granted under the below condition:

1. If the request made by the process is less than or equal to max need to that process.
2. If the request made by the process is less than or equal to the freely available resource in the system.

The algorithm for finding out whether or not a system is in a safe state:

1) Let Work and Finish be vectors of length 'm' and 'n' respectively.

Initialize: Work = Available

Finish[i] = false; for i=1, 2, 3, 4….n

2) Find an i such that both

a) Finish[i] = false

b) Needi <= Work

if no such i exists goto step (4)

3) Work = Work + Allocation[i]

Finish[i] = true

goto step (2)

4) if Finish [i] = true for all i

then the system is in a safe state

Resource-Request Algorithm:

1) If Requesti <= Needi

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If Requesti <= Available

Goto step (3); otherwise, Pi must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process Pi by modifying the state as

follows:

Available = Available – Requesti

Allocationi = Allocationi + Requesti

Needi = Needi– Requesti

4) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred. If resources have a single instance then a cycle would be enough to check for a deadlock as shown in fig 1, but not in multiple instances of resources.
There are 2 methods to resolve it:

1. **Killing the process:** Kill the process one by one involved in deadlock and check the system and continue this till the system is out of deadlock.
2. **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, which then allocated to other processes so that there is a possibility of recovering the system from deadlock but the system can go into starvation.

Deadlock Detection Algorithm:

1. Let *Work* and *Finish* be vectors of length m and n respectively. Initialize *Work= Available*. For *i=0, 1, …., n-1*, if *Requesti* = 0, then *Finish[i]* = true; otherwise, *Finish[i]*= false.
2. Find an index i such that both
   a) *Finish[i] == false*
   b) *Requesti <= Work*
   If no such *i* exists go to step 4.
3. *Work= Work + Allocationi*
   Finish[i]= true
   Go to Step 2.
4. If *Finish[i]== false* for some i, 0<=i<n, then the system is in a deadlocked state. Moreover, if *Finish[i]==false* the process Pi is deadlocked.

A system has R identical resources, P processes competing for them and N is the maximum need of each process. The minimum no.of resources required so that deadlock will never occur.

Formula:

```
R >= P * (N - 1) + 1
```

In a distributed system deadlock can neither be prevented nor avoided as the system is so vast that it is impossible to do so. Therefore, only deadlock detection can be implemented. The techniques of deadlock detection in the distributed system require the following:

- **Progress –**
  The method should be able to detect all the deadlocks in the system.
- **Safety –**
  The method should not detect false or phantom deadlocks.

There are three approaches to detect deadlocks in distributed systems**:-**

1.Centralized approach –

Only one responsible resource is there to detect deadlock.
Simple and easy to implement.
Excessive workload at one node, single-point failure which in turn makes the system less reliable.

In this two techniques are used**:-**

- **Completely Centralized Algorithm –**
  In a network of n sites, one site is chosen as a control site which is responsible for deadlock detection and has control over all resources. If a site requires a resource it requests the control site, it allocates and deallocates resources and maintains a wait for graph. And at a regular interval of time, it checks the wait for the graph to detect a cycle. If a cycle exists then it will declare the system as deadlock otherwise the system will continue working. The major drawbacks of this technique are as follows:
  1. A site has to send a request even for using its own resource.
  2. There is a possibility of phantom deadlock.
- **HO Ramamurthy (Two-Phase Algorithm) –**
  In this technique a resource status table is maintained by the central or control site, if a cycle is detected then the system is not declared deadlock at first, the cycle is checked again as the system is distributed some or the other resource is vacant or freed by sites at every instant of time. Now, after checking if a cycle is detected again then, the system is declared as

deadlock. Itreduces the possibility of phantom deadlock but on the other hand time consumption is more.

- **HO Ramamurthy (One Phase Algorithm) –**
  In this technique a resource status table and a process table is maintained by the central or control site if the cycle is detected in both processes and resource tables then the system is declared as deadlock. It reduces time consumption but space complexity increases.

2.Distributed approach –

Different nodes work together to detect deadlocks.
No single point failure as the workload is equally divided among all nodes.
The speed of deadlock detection also increases.

3.Hierarchical approach –
This approach is the most advantageous. It is the combination of both centralized and distributed approaches. In this approach, some selected nodes or clusters of nodes are responsible for deadlock detection and these selected nodes are controlled by a single node.

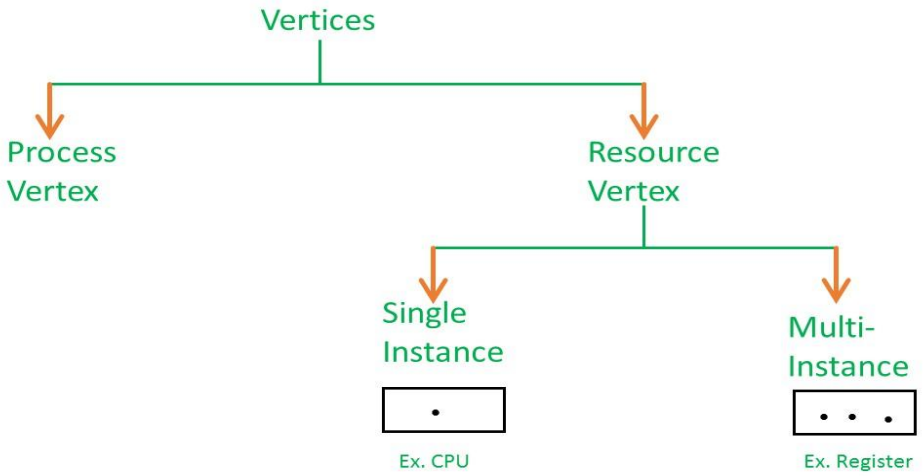
Difference between Deadlock, Starvation, and Livelock

A livelock is similar to a deadlock, except that the states of the processes involved in the livelock constantly change with regard to one another, none progressing. Livelock is a special case of resource starvation; the general definition only states that a specific process is not progressing.
Starvation is a problem which is somewhat related to both, Livelock and Deadlock. In a dynamic system, requests for resources keep on happening. Thereby, some policy is needed to make a decision about who gets the resource when. This process, being reasonable, may lead to some processes never getting serviced even though they are not deadlocked.

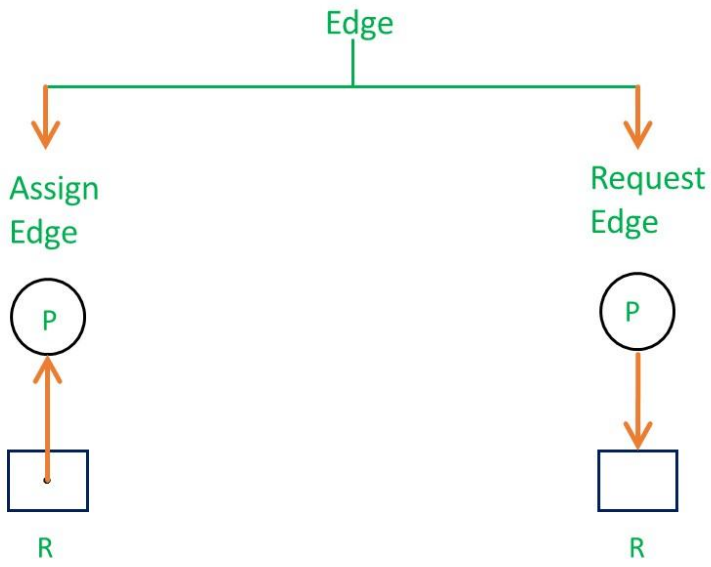
Resource Allocation Graph (RAG)

It explains the state of the system in terms of **processes and resources** i.e.how many resources are available, how many are allocated and what is the request of each process.

In RAG vertices are of two types –



In RAG edges are of two types –



	Resource partitioning approach	Pool based approach
Definition	OS decides beforehand, what resources should be	OS checks the allocation status in the resource

	allocated to which user program and divides them in the system to many *resource partitions*, where each partition may include various resources. Then, it allocates one resource partition to each user program before the program's initiation.	table whenever a program makes a request for a resource. If the resource is free, it allocates the resource to the program.
Advantages	Easy to implement. Less Overhead.	Allocated resources are not wasted. Any resource requirement can be fulfilled if the resource is free (unlike Partitioning approach)
Disadvantages	Lacks flexibility	Overhead of allocating and deallocating the resources on every request and release.