**UNIT – III**

**Deadlocks** - System Model, Deadlocks Characterization, Methods for Handling Deadlocks, Deadlock Prevention, Deadlock Avoidance, Deadlock Detection, and Recovery from Deadlock Process.

**Management and Synchronization** - The Critical Section Problem, Synchronization Hardware, Semaphores, and Classical Problems of Synchronization, Critical Regions, Monitors Interprocess Communication Mechanisms: IPC between processes on a single computer system, IPC between processes on different systems, using pipes, FIFOs, message queues, shared memory.

**Introduction to Deadlocks**

***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.

For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



**3.1 System Model**

A system model or structure consists of a fixed number of resources to be circulated among some opposing processes. The resources are then partitioned into numerous types, each consisting of some specific quantity of identical instances. Such as Memory space, CPU cycles, directories and files, I/O devices like keyboards, printers and CD-DVD drives are prime examples of resource types. When a system has 2 CPUs, then the resource type CPU got two instances.

Under the standard mode of operation, any process may use a resource in only the below-mentioned sequence:

1. **Request:** When the request can't be approved immediately, then the requesting job must remain waited until it can obtain the resource.
2. **Use:** The process can run on the resource.
3. **Release:** The process releases the resource.

**Difference between Starvation and Deadlock**

|  |  |  |
| --- | --- | --- |
| **Sr.** | **Deadlock** | **Starvation** |
| **1** | **Deadlock is a situation where no process got blocked and no process proceeds** | **Starvation is a situation where the low priority process got blocked and the high priority processes proceed.** |
| **2** | **Deadlock is an infinite waiting.** | **Starvation is a long waiting but not infinite.** |
| **3** | **Every Deadlock is always a starvation.** | **Every starvation need not be deadlock.** |
| **4** | **The requested resource is blocked by the other process.** | **The requested resource is continuously be used by the higher priority processes.** |
| **5** | **Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.** | **It occurs due to the uncontrolled priority and resource management.** |

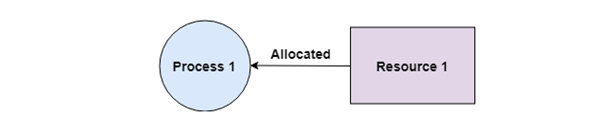
**3.2 Deadlocks Characterization**

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

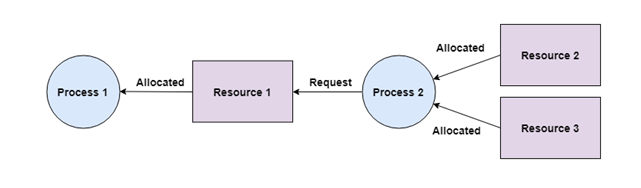
There are four conditions that are necessary to achieve deadlock:

1. **Mutual Exclusion**

A resource can only be shared in mutually exclusive manner. It implies, if two processes cannot use the same resource at the same time.

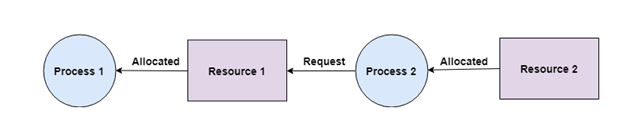


1. **Hold and Wait**

A process waits for some resources while holding another resource at the same time. 

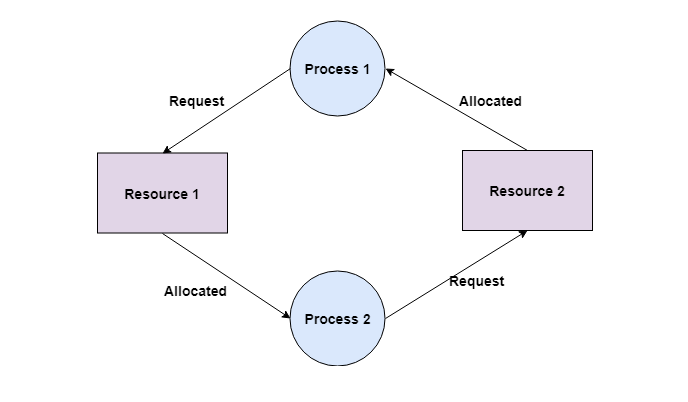
1. **No preemption**

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.



1. **Circular Wait**

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.



**3.3.** **Methods for Handling Deadlocks**

1. Deadlock Prevention
2. Deadlock Avoidance
3. Deadlock Detection
4. Deadlock Recovery

**3.3.1 Deadlock Prevention**

**Mutual Exclusion** – not required for sharable resources; must hold for non-sharable resources.

**Hold and Wait –** must guarantee that whenever a process requests a resource, it does not hold any other resources.

**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.

**3.3.2 Deadlock Avoidance**

In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs. The process continues until the system is in safe state. Deadlock avoidance can be done with Banker’s Algorithm.

**Banker’s Algorithm**

Bankers’ 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.

**Safety Algorithm**

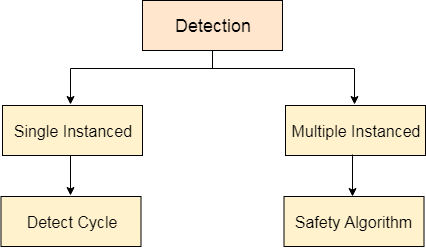
The algorithm for finding out whether or not a system is in a safe state can be described as follows:

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*

**Example:**

**3.3.3 Deadlock Detection**

The Operating System can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graphs, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes.

**Preempt the resource**

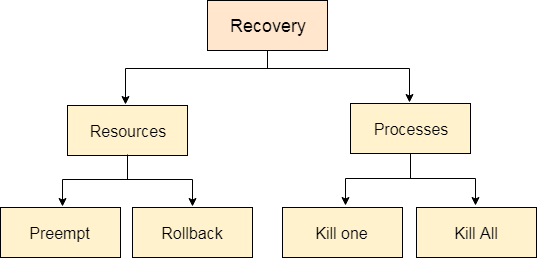
We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

**Rollback to a safe state**

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

**3.3.4 Deadlock Recovery**

In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.



* **For Resource**

**Preempt the resource**

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner.

**Rollback to a safe state**

System passes through various states to get into the deadlock state. The operating systems can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state. We will rollback all the allocations to get into the previous safe state.

* **For Process**

**Kill a process**

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

**Kill all process**

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.