Deadlock

Deadlock: Is it a state where two ore more operations are waiting for each other, say a computing action 'A' is waiting for action 'B' to complete, while action 'B' can only execute when 'A' is completed. Such a situation would be called a deadlock. In operating systems, a deadlock situation is arrived when computer resources required for complete of a computing task are held by another task that is waiting to execute. The system thus goes into an indefinite loop resulting into a deadlock.The deadlock in operating system seems to be a common issue in multiprocessor systems, parallel and distributed computing setups.

The resources may be either physical or logical. Examples of physical resources are Printers, Tape Drivers, Memory Space, and *CPU* Cycles. Examples of logical resources are Files, Semaphores, and Monitors.The simplest example of deadlock is where process 1 has been allocated non-shareable resources *A*, say, a tap drive, and process 2 has be allocated non-sharable resource *B*, say, a printer. Now, if it turns out that process 1 needs resource *B* (printer) to proceed and process 2 needs resource *A* (the tape drive) to proceed and these are the only two processes in the system, each is blocked the other and all useful work in the system stops. This situation ifs termed deadlock. The system is in deadlock state because each process holds a resource being requested by the other process neither process is willing to release the resource it holds.Resources come in two flavors: preemptable and non preemptable. A preemptable resource is one that can be taken away from the process with no ill effects. Memory is an example of a preemptable resource. On the other hand, a non preemptable resource is one that cannot be taken away from process (without causing ill effect). For example, *CD* resources are not preemptable at an arbitrary moment.Reallocating resources can resolve deadlocks that involve preemptable resources.

Deadlocks that involve non preemptable resources are difficult to deal with.

In order for deadlock to occur, four conditions must be true.

- **Mutual exclusion** – Each resource is either currently allocated to exactly one process or it is available. (Two processes cannot simultaneously control the same resource or be in their critical section).
- **Hold and Wait** – processes currently holding resources can request new resources
- **No preemption** – Once a process holds a resource, it cannot be taken away by another process or the kernel.
- **Circular wait** – Each process is waiting to obtain a resource which is held by another process.

Following three strategies can be used to remove deadlock after its occurrence:

1. **Preemption**We can take a resource from one process and give it to other. This will resolve the deadlock situation, but sometimes it does causes problems.
2. **Rollback**In situations where deadlock is a real possibility, the system can periodically make a record of the state of each process and when deadlock occurs, roll everything back to the last checkpoint, and restart, but allocating resources differently so that deadlock does not occur.
3. **Kill one or more processes**This is the simplest way, but it works.

***Livelock**: A situation in which two or more processes continuously change their states in response to changes in the other process(es) without doing any useful work. It is somewhat similar to the deadlock but the difference is processes are getting polite and let other to do the work. This can be happen when a process trying to avoid a deadlock.*

Dijkstra Banking Algorithm :

The Banker's Algorithm is a strategy for **deadlock** prevention. In an operating system, deadlock is a state in which two or more processes are "stuck" in a **circular wait** state. All deadlocked processes are waiting for resources held by other processes. Because most systems are **non-preemptive** (that is, will not take resources held by a process away from it), and employ a **hold and wait** method for dealing with system resources (that is, once a process gets a certain resource it will not give it up voluntarily), deadlock is a dangerous state that can cause poor system performance.

One reason this algorithm is not widely used in the real world is because to use it the operating system must know the maximum amount of resources that every process is going to need at all times. Therefore, for example, a just-executed program must declare up-front that it will be needing no more than, say, 400K of memory. The operating system would then store the limit of 400K and use it in the deadlock avoidance calculations.The Banker's Algorithm seeks to prevent deadlock by becoming involved in the granting or denying of system resources. Each time that a process needs a particular **non-sharable** resource, the request must be approved by the banker.