**DEADLOCK PREVENTION**

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

**1)Mutual Exclusion:**

In terms of resources, mutual exclusion means that multiple processes can’t use the same resource at the same time.  For example, a printer cannot be simultaneously shared by several processes. In contrast, Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock.  A good example of a sharable resource is Read-only files because if several processes attempt to open a read-only file at the same time, then they can be granted simultaneous access to the file.

A process never needs to wait for a sharable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically non-sharable.

**2)Hold and Wait:**

A hold and wait condition occurs when a process holds a resource while waiting for other resources to complete its task.

**!(Hold and wait) = !hold or !wait (the negation of hold and wait is that either one doesn’t hold or they don’t wait)**

One protocol that can be used requires each process to request and be allocated all its resources before it begins execution. The process is then started without having to wait for any resources. In practice, we can only do so if we first identify all of the resources that the process will require. Although this sounds highly practical, we cannot do so in a computer system because no process can identify the required resources at the outset.

An alternative protocol allows a process to request resources only when it has it does not occupy any resource. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

To illustrate the difference between these two protocols, we consider a process that copies data from a DVD drive to a file on disk, sorts the file, and then prints the results to a printer. If all resources must be requested at the beginning of the process, then the process must initially request the DVD drive, disk file, and printer. It will hold the printer for its entire execution, even though it needs the printer only at the end.

The second method allows the process to request initially only the DVD drive and disk file. It copies from the DVD drive to the disk and then releases both the DVD drive and the disk file. The process must then request the disk file and the printer. After copying the disk file to the printer, it releases these two resources and terminates.

Both these protocols have two main disadvantages. First, resource utilization may be low, since resources may be allocated but unused for a long period. In the example given, for instance, we can release the DVD drive and disk file, and then request the disk file and printer, only if we can be sure that our data will remain on the disk file. Otherwise, we must request all resources at the beginning for both protocols.

Second, starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

**3)No Preemption:**

The third necessary condition for deadlocks is that there be no pre-emption of resources that have already been allocated. To ensure that this condition does not hold, we can use the following protocol. If a process is holding some resources and requests another resource that cannot be immediately allocated to it (that is, the process must wait), then all resources the process is currently holding are preempted. In other words, these resources are implicitly released. The preempted resources are added to the list of resources for which the process is waiting. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Alternatively, if a process requests some resources, we first check whether they are available. If they are, we allocate them. If they are not, we check whether they are allocated to some other process that is waiting for additional resources. If so, we preempt the desired resources from the waiting process and allocate them to the requesting process. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were pre-empted while it was waiting.

**4)Circular Wait:**

The fourth and final condition for deadlocks is the circular-wait condition. A circular wait occurs when one or more processes wait in a circular order for the resources they require. In order to ensure violate this condition we can do the following:  Assign a priority number to each resource. There will be a condition that a process can request the resources to increase/decrease. order of numbering, that means any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.

Example: Assume that R5 resource is allocated to P1, if next time P1 asks for R4, R3 that are lesser than R5; then such request will not be granted. Only the request for resources that are more than R5 will be granted.