Deadlock

What is Deadlock ?

A deadlock is a situation when a process in the system has acquired some resources and waiting for more resources which are acquired by some other process which in turn is waiting for the resources acquired by this process. Hence, none of them can proceed and OS cant do any work.

In an operating system, a **deadlock** occurs when a process or thread enters a waiting state because a requested system resources is held by another waiting process, which in turn is waiting for another resource held by another waiting process. If a process is unable to change its state indefinitely because the resources requested by it are being used by another waiting process, then the system is said to be in a deadlock.

Necessary Conditions

A deadlock situation on a resource can arise if and only if all of the following conditions hold simultaneously in a system:

- **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. A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
- **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.

These four conditions are known as the *Coffman conditions*.

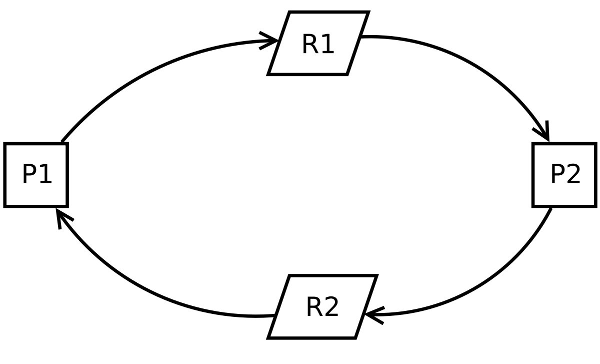

Figure – 1

The figure – 1 what we see above : Both processes need resources to continue execution. *P1* requires additional resource *R1* and is in possession of resource *R2*, *P2* requires additional resource *R2* and is in possession of *R1*; neither process can continue.

Deadlock handling

Most current operating systems cannot prevent deadlocks.When a deadlock occurs, different operating systems respond to them in different non-standard manners. Most approaches work by preventing one of the four Coffman conditions from occurring, especially the fourth one. Major approaches are as follows :

Ignoring deadlock : In this approach, it is assumed that a deadlock will never occur. This is also an application of the Ostrich algorithm. This approach was initially used by MINIX and UNIX.This is used when the time intervals between occurrences of deadlocks are large and the data loss incurred each time is tolerable.

Detection : Under the deadlock detection, deadlocks are allowed to occur. Then the state of the system is examined to detect that a deadlock has occurred and subsequently it is corrected. An algorithm is employed that tracks resource allocation and process states, it rolls back and restarts one or more of the processes in order to remove the detected deadlock. Detecting a deadlock that has already occurred is easily possible since the resources that each process has locked and/or currently requested are known to the resource scheduler of the operating system.

After a deadlock is detected, it can be corrected by using one of the following methods:

- *Process termination :* one or more processes involved in the deadlock may be aborted. One could choose to abort all competing processes involved in the deadlock. This ensures that deadlock is resolved with certainty and speed. But the expense is high as partial computations will be lost. Or, one could choose to abort one process at a time until the deadlock is resolved. This approach has high overhead because after each abort an algorithm must determine whether the system is still in deadlock. Several factors must be considered while choosing a candidate for termination, such as priority and age of the process.

- *Resource preemption :* resources allocated to various processes may be successively preempted and allocated to other processes until the deadlock is broken.

Deadlock prevention algorithms

In computer science, **deadlock prevention algorithms** are used in concurrent programming when multiple processes must acquire more than one shared resource. If two or more concurrent processes obtain multiple resources indiscriminately, a situation can occur where each process has a resource needed by another process. As a result, none of the processes can obtain all the resources it needs, so all processes are blocked from further execution. This situation is called a deadlock. A deadlock prevention algorithm organizes resource usage by each process to ensure that at least one process is always able to get all the resources it needs.

The Next Section I will try to Cover “Banker’s Algorithm”