Deadlock

In the multiprogramming operating system, there are a number of processing which fights for a finite number of resources and sometimes waiting process never gets a chance to change its state because the resources for which it is waiting are held by another waiting process. A set of a process is called deadlock when they are waiting for the happening of an event which is called by some another event in the same set.

Here every process will follow the system model which means the process requests a resource if not allocated then wait otherwise it allocated will use the resources and release it after use.

Methods for handling deadlock

There are mainly four methods for handling deadlock.

1. Deadlock ignorance

It is the most popular method and it acts as if no deadlock and the user will restart. As handling deadlock is expensive to be called of a lot of codes need to be altered which will decrease the performance so for less critical jobs deadlock are ignored. Ostrich algorithm is used in deadlock Ignorance. Used in windows, Linux etc.

2. Deadlock prevention

It means that we design such a system where there is no chance of having a deadlock.

* **Mutual exclusion:**  
  It can’t be resolved as it is the hardware property. For example, the printer cannot be simultaneously shared by several processes. This is very difficult because some resources are not sharable.
* **Hold and wait:**  
  Hold and wait can be resolved using the conservative approach where a process can start it and only if it has acquired all the resources.
* **Active approach:**  
  Here the process acquires only requires resources but whenever a new resource requires it must first release all the resources.
* **Wait time out:**  
  Here there is a maximum time bound until which a process can wait for other resources after which it must release the resources.
* **Circular wait:**  
  In order to remove circular wait, we assign a number to every resource and the process can request only in the increasing order otherwise the process must release all the high number acquires resources and then make a fresh request.
* **No pre-emption:**  
  In no pre-emption, we allow forceful pre-emption where a resource can be forcefully pre-empted. The pre-empted resource is added to the list of resources where the process is waiting. The new process can be restarted only when it regains its old resources. Priority must be given to a process which is in waiting for state.

3. Deadlock avoidance

Here whenever a process enters into the system it must declare maximum demand. To the deadlock problem before the deadlock occurs. This approach employs an algorithm to access the possibility that deadlock would occur and not act accordingly. If the necessary condition of deadlock is in place it is still possible to avoid feedback by allocating resources carefully.

**A deadlock avoidance algorithm** dynamically examines the resources allocation state to ensure that a circular wait condition case never exists. Where the resources allocation state is defined by the of available and allocated resources and the maximum demand of the process. There are 3 states of the system:

**Safe state**

When a system can allocate the resources to the process in such a way so that they still avoid deadlock then the state is called safe state. When there is a safe sequence exit then we can say that the system is in the safe state.

A sequence is in the safe state only if there exists a safe sequence. A sequence of process **P1, P2, Pn** is a safe sequence for the current allocation state if for each **Pi** the resources request that **Pi** can still make can be satisfied by currently available resources pulls the resources held by all **Pj** with **j<i**.

**Methods for deadlock avoidance**

**1) Resource allocation graph**

This graph is also kind of graphical bankers' algorithm where a process is denoted by a circle Pi and resources is denoted by a rectangle RJ (.dots) inside the resources represents copies.

Presence of a cycle in the resources allocation graph is necessary but not sufficient condition for detection of deadlock. If the type of every resource has exactly one copy than the presence of cycle is necessary as well as sufficient condition for detection of deadlock.

This is in unsafe state (cycle exist) if P1 request P2 and P2 request R1 then deadlock will occur.

**2) Bankers’s algorithm**

The resource allocation graph algorithms not applicable to the system with multiple instances of the type of each resource. So for this system Banker’s algorithm is used.

Here whenever a process enters into the system it must declare maximum demand possible.