DEPARTMENT OF AIML

23CST202- OPERATING SYSTEMS

II YEAR IV SEM AIML-B

UNIT 2-PROCESS SCHEDULING AND SYNCHRONIZATION

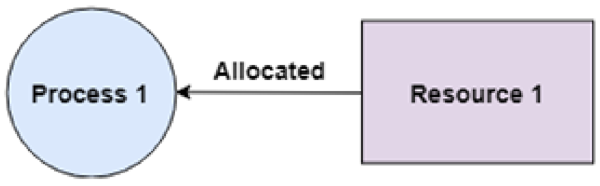
TOPIC –DEADLOCK CHARACTERIZATION,METHODS OF HANDLING DEADLOCK

DEADLOCK 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.
A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

Mutual Exclusion

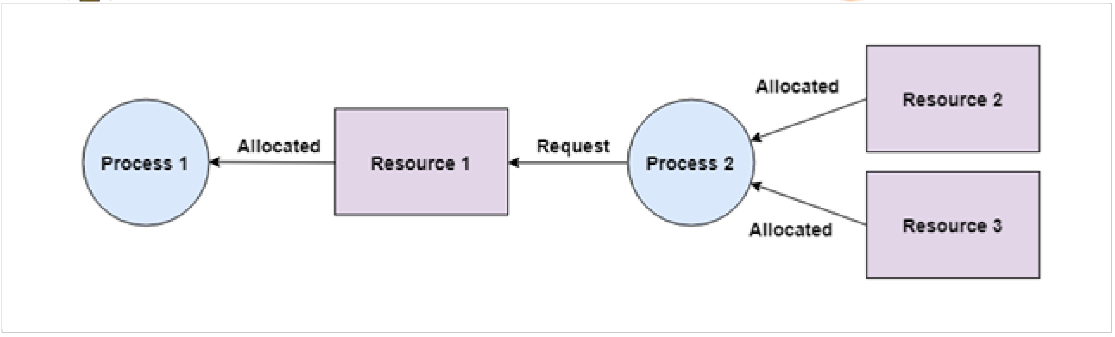
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 oXnly.



.

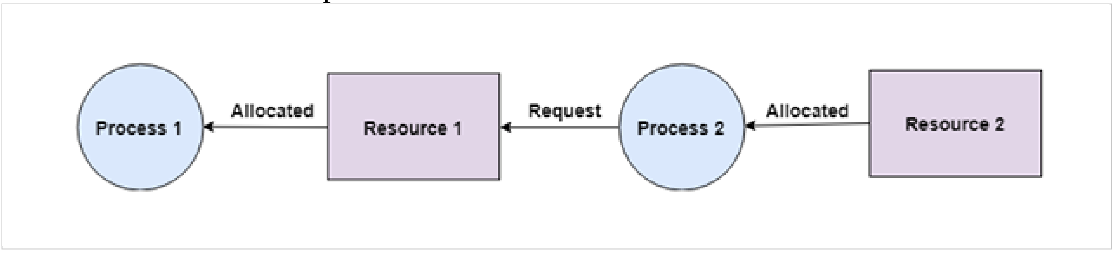
Hold and Wait

A **process** can hold multiple resources and still request more res.Nources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



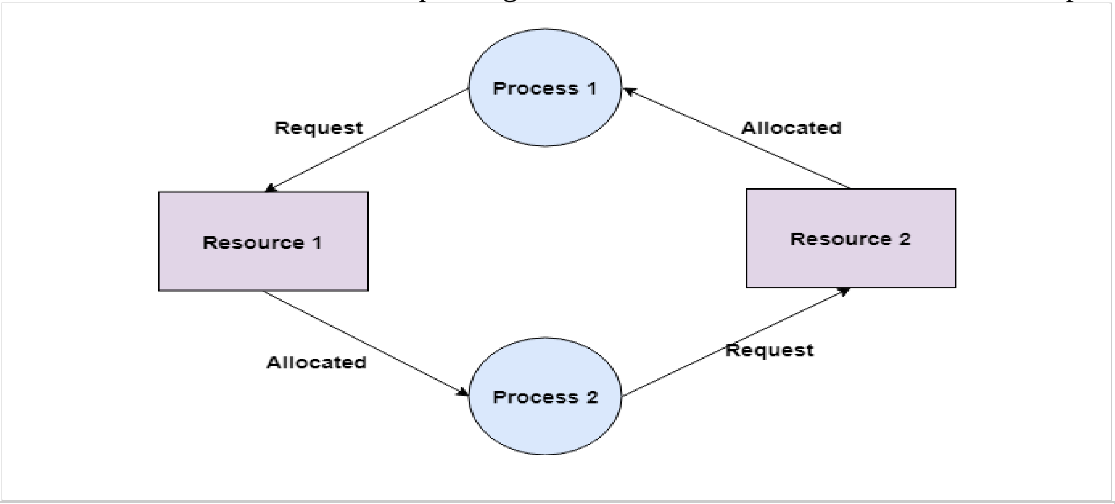
No Preemption

A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



METHODS OF HANDLING DEADLOCK

Deadlock is a situation where a process or a set of processes is blocked, waiting for some other resource that is held by some other waiting process. It is an undesirable state of the system. In other words, Deadlock is a critical situation in computing where a process, or a group of processes, becomes unable to proceed because each is waiting for a resource that is held by another process in the same group. This scenario leads to a complete standstill, rendering the affected processes inactive and the system inefficient.

Necessary Condition for a Deadlock
The following are the four conditions that must hold simultaneously for a deadlock to occur.

1. **Mutual Exclusion:** A resource can be used by only one process at a time. If another process requests for that resource, then the requesting process must be delayed until the resource has been released.
2. **Hold and wait:** Some processes must be holding some resources in the non-shareable mode and at the same time must be waiting to acquire some more resources, which are currently held by other processes in the non-shareable mode.
3. **No pre-emption:** Resources granted to a process can be released back to the system only as a result of voluntary action of that process after the process has completed its task.
4. **Circular wait:** Deadlocked processes are involved in a circular chain such that each process holds one or more resources being requested by the next process in the chain.

Methods of Handling Deadlocks
There are four approaches to dealing with deadlocks.

1. Deadlock Prevention
2. Deadlock avoidance (Banker's Algorithm)
3. Deadlock detection & recovery
4. Deadlock Ignorance (Ostrich Method)

These are explained below.

1. Deadlock Prevention
The strategy of deadlock prevention is to design the system in such a way that the possibility of deadlock is excluded. The indirect methods prevent the occurrence of one of three necessary conditions of deadlock i.e., mutual exclusion, no pre-emption, and hold and wait. The direct method prevents the occurrence of circular wait. **Prevention techniques -Mutual exclusion** – are supported by the OS. **Hold and Wait** – the condition can be prevented by requiring that a process requests all its required resources at one time and blocking the process until all of its requests can be granted at the same time simultaneously. But this prevention does not yield good results because:

- long waiting time required
- inefficient use of allocated resource
- A process may not know all the required resources in advance

No pre-emption – techniques for 'no pre-emption are'

- If a process that is holding some resource, requests another resource that can not be immediately allocated to it, all resources currently being held are released and if necessary, request again together with the additional resource.
- If a process requests a resource that is currently held by another process, the OS may pre-empt the second process and require it to release its resources. This works only if both processes do not have the same priority.

Circular wait One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in increasing order of enumeration, i.e., if a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in ordering.

2. Deadlock Avoidance

The deadlock avoidance Algorithm works by proactively looking for potential deadlock situations before they occur. It does this by tracking the resource usage of each process and identifying conflicts that could potentially lead to a deadlock. If a potential deadlock is identified, the algorithm will take steps to resolve the conflict, such as rolling back one of the processes or pre-emptively allocating resources to other processes. The Deadlock Avoidance Algorithm is designed to minimize the chances of a deadlock occurring, although it cannot guarantee that a deadlock will never occur. This approach allows the three necessary conditions of deadlock but makes judicious choices to assure that the deadlock point is never reached. It allows more concurrency than avoidance detection A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to deadlock. It requires knowledge of future process requests. Two techniques to avoid deadlock :

1. Process initiation denial
2. Resource allocation denial

Advantages

- Not necessary to pre-empt and rollback processes
- Less restrictive than deadlock prevention

Disadvantages

- Future resource requirements must be known in advance
- Processes can be blocked for long periods
- Exists a fixed number of resources for allocation

Banker's Algorithm

The Banker's Algorithm is based on the concept of resource allocation graphs. A resource allocation graph is a directed graph where each node represents a process, and each edge represents a resource. The state of the system is represented by the current allocation of resources between processes. For example, if the system has three processes, each of which is using two resources, the resource allocation graph would look like this:

Processes A, B, and C would be the nodes, and the resources they are using would be the edges connecting them. The Banker's Algorithm works by analyzing the state of the system and determining if it is in a safe state or at risk of entering a deadlock.

To determine if a system is in a safe state, the Banker's Algorithm uses two matrices: the available matrix and the need matrix. The available matrix contains the amount of each resource currently available. The need matrix contains the amount of each resource required by each process.

The Banker's Algorithm then checks to see if a process can be completed without overloading the system. It does this by subtracting the amount of each resource used by the process from the available matrix and adding it to the need matrix. If the result is in a safe state, the process is allowed to proceed, otherwise, it is blocked until more resources become available.

The Banker's Algorithm is an effective way to prevent deadlocks in multiprogramming systems. It is used in many operating systems, including Windows and Linux. In addition, it is used in many other types of systems, such as manufacturing systems and banking systems.

The Banker's Algorithm is a powerful tool for resource allocation problems, but it is not foolproof. It can be fooled by processes that consume more resources than they need, or by processes that produce more resources than they need. Also, it can be fooled by processes that consume resources in an unpredictable manner. To prevent these types of problems, it is important to carefully monitor the system to ensure that it is in a safe state.

3. Deadlock Detection

Deadlock detection is used by employing an algorithm that tracks the circular waiting and kills one or more processes so that the deadlock is removed. The system state is examined periodically to determine if a set of processes is deadlocked. A deadlock is resolved by aborting and restarting a process, relinquishing all the resources that the process held.

- This technique does not limit resource access or restrict process action.
- Requested resources are granted to processes whenever possible.
- It never delays the process initiation and facilitates online handling.
- The disadvantage is the inherent pre-emption losses.

4.Deadlock Ignorance

In the Deadlock ignorance method the OS acts like the deadlock never occurs and completely ignores it even if the deadlock occurs. This method only applies if the deadlock occurs very rarely. The algorithm is very simple. It says, " if the deadlock occurs, simply reboot the system and act like the deadlock never occurred." That's why the algorithm is called the **Ostrich Algorithm**.

Advantages

- Ostrich Algorithm is relatively easy to implement and is effective in most cases.
- It helps in avoiding the deadlock situation by ignoring the presence of deadlocks.

Disadvantages

- Ostrich Algorithm does not provide any information about the deadlock situation.
- It can lead to reduced performance of the system as the system may be blocked for a long time.