UNIT III

Deadlocks

Deadlock:

A deadlock occurs when a set of processes is stalled because each process is holding a resource and waiting for another process to acquire another resource. In the diagram below, for example, Process 1 is holding Resource 1 while Process 2 acquires Resource 2, and Process 2 is waiting for Resource 1.

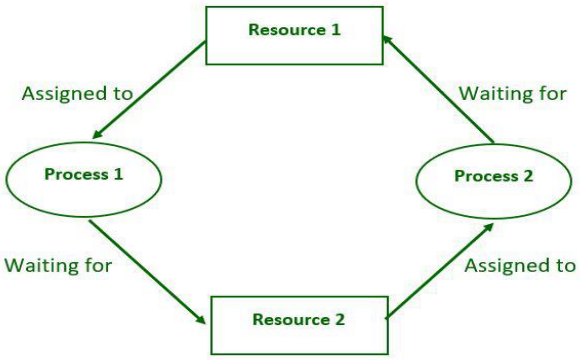


Figure: Deadlock in Operating system

System Model:

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources that can be divided into different categories and allocated to a variety of processes, each with different requirements.
- Memory, printers, CPUs, open files, tape drives, CD-ROMs, and other resources are examples of resource categories.
- By definition, all resources within a category are equivalent, and any of the resources within that category can equally satisfy a request from that category. If this is not the case (i.e. if there is some difference between the resources within a category), then that category must be subdivided further.

For example: the term “printers” may need to be subdivided into “laser printers” and “color inkjet printers.”

- Some categories may only have one resource.
- The kernel keeps track of which resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available for all kernel-managed resources.
- Mutexes or wait() and signal() calls can be used to control application-managed resources (i.e. binary or counting semaphores.)
- When every process in a set is waiting for a resource that is currently assigned to another process in the set, the set is said to be deadlocked.

Operations:

1. **Request**

   If the request cannot be granted immediately, the process must wait until the resource(s) required to become available. The system, for example, uses the functions open(), malloc(), new(), and request ().
2. **Use**

   The process makes use of the resource, such as printing to a printer or reading from a file.
3. **Release**

   The process relinquishes the resource, allowing it to be used by other processes.

Deadlock Characterization:

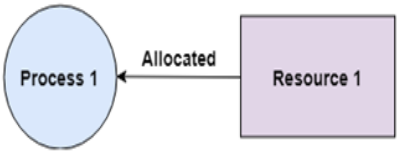
A deadlock situation can arise if the following four conditions hold simultaneously in a system:

1. Mutual exclusion.
2. Hold and wait.
3. No preemption.

4. Circular wait.

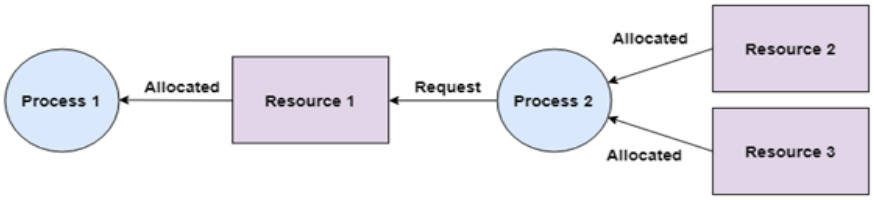
Mutual Exclusion

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 only.



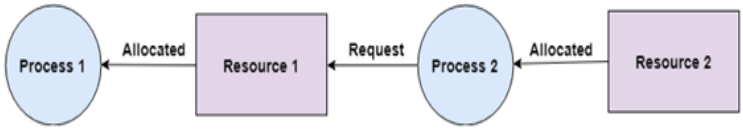
Hold and Wait

A process can hold multiple resources and still request more resources 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

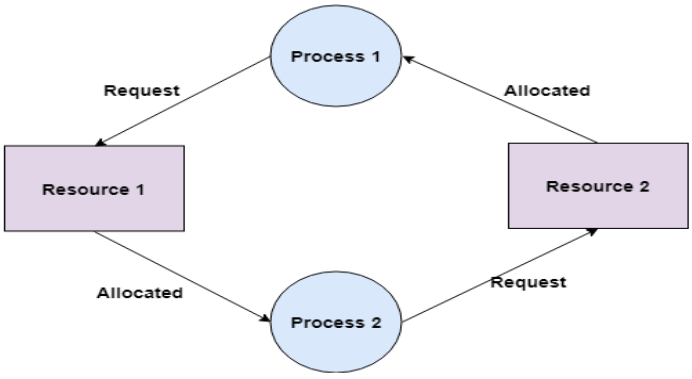
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



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 for Handling Deadlocks:

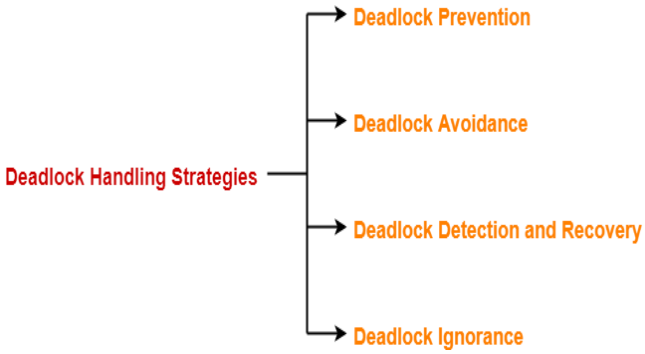
There are four approaches to dealing with deadlocks.

1. Deadlock Prevention

2. Deadlock avoidance (Banker's Algorithm)

3. Deadlock detection & recovery

4. Deadlock Ignorance (Ostrich Method)



Deadlock Prevention:

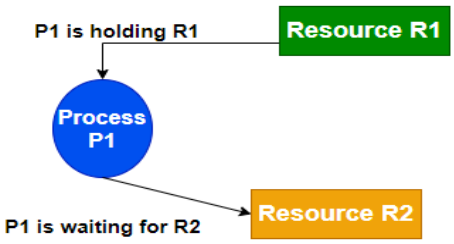
The strategy of deadlock prevention is to design the system in such a way that the possibility of deadlock is excluded. The indirect methods prevent the occurrence of one of three necessary conditions of deadlock .

1. **Prevention techniques & Mutual exclusion –** are supported by the OS.
2. **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.



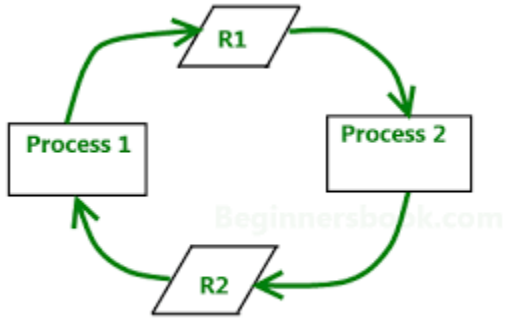
Hold and wait condition

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 :
  a) Process initiation denial
  b) Resource allocation denial
- **Advantages of deadlock avoidance techniques:**
  a) Not necessary to pre-empt and rollback processes
  b) Less restrictive than deadlock prevention
- **Disadvantages :**
  a) Future resource requirements must be known in advance

b) Processes can be blocked for long periods

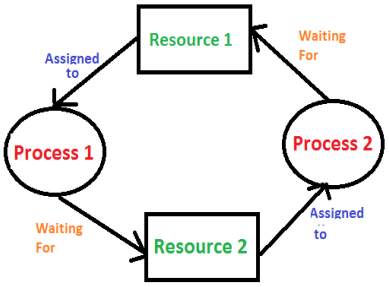
c) Exists a fixed number of resources for allocation



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.



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.
- It can lead to a resource leak, as resources are not released when the system is blocked due to deadlock.

.