Deadlock

Explain deadlock and its necessary conditions.
Describe various methods of handling deadlock.

Condition when deadlock occurs

Mutual Exclusion : It makes sure that one process can hold only one resource at a time

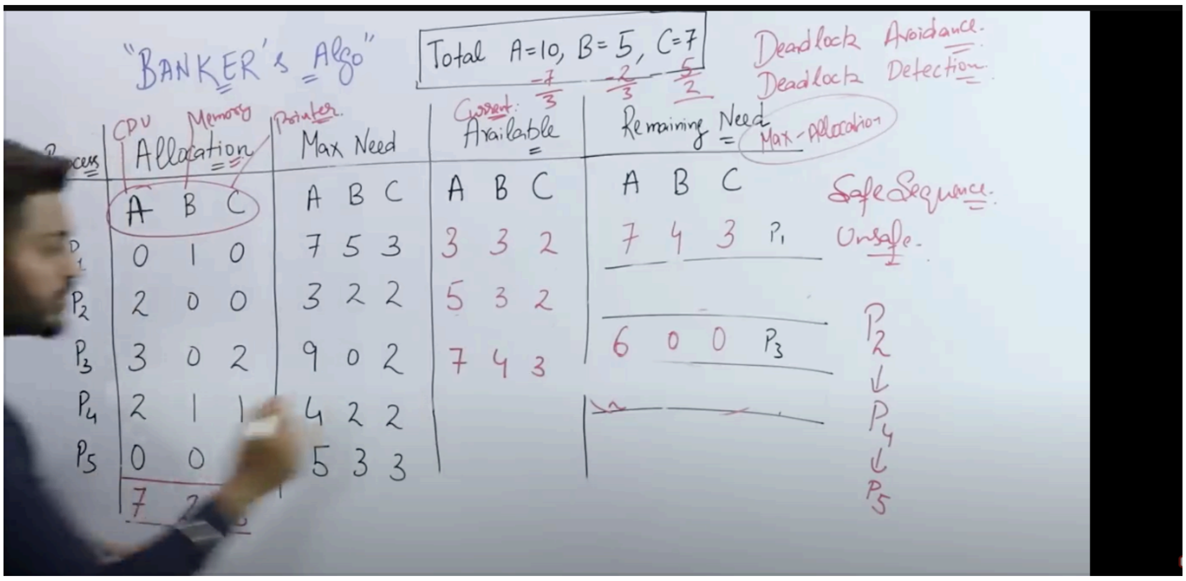
Pre-emption process: -> Means you can't take a resource from a process until it used it completely
To prevent It makes sure that when two process stuck for each other we must complete one of them first so that other can use that resource after on.
And to do this we will simply send one of them to ready queue so that its resources gets free for other one.

Hold & Wait : A process must be holding a resource while waiting for another resource

Circular Wait: Multiple processes waiting for resources in a circular loop

Avoidance of Deadlock
1. Deadlock Ignorance
2. Deadlock avoidance
3. Deadlock prevention
4. Deadlock detect and correction

Banker's Algorithm for deadlock avoidance
-> deadlock only occurs when we won't be able to fulfill remaining need for any of the processor
-> here the system needs pre information of the
-> in real life this is hard to implement because of the max need's static data, I mean we can't tell how how resources a processor might need in future exactly !



"BANKER's Algo"

Total A=10, B=5, C=7

-7 / 3, -2 / 3, -5 / 2

Deadlock Avoidance.

Deadlock Detection.

Process	Allocation (CPU Memory Printer)			Max Need			Current Available			Remaining Need = Max - Allocation			
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	0	1	0	7	5	3	3	3	2	7	4	3	P1
P2	2	0	0	3	2	2	5	3	2				
P3	3	0	2	9	0	2	7	4	3	6	0	0	P3
P4	2	1	1	4	2	2							
P5	0	0	[illegible]	5	3	3							
	7	2	[illegible]										

Safe Sequence.

Unsafe.

P2 ↓ P4 ↓ P5

What is Deadlock?

Deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources. Key concepts include mutual exclusion, resource holding, circular wait, and no preemption. Consider a practical example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.

Deadlock is an infinite Process it means that once a process goes into deadlock it will never come out of the loop and the process will enter for an indefinite amount of time. There are only detection, resolution, and prevention techniques. But, there are no Deadlock-stopping techniques.

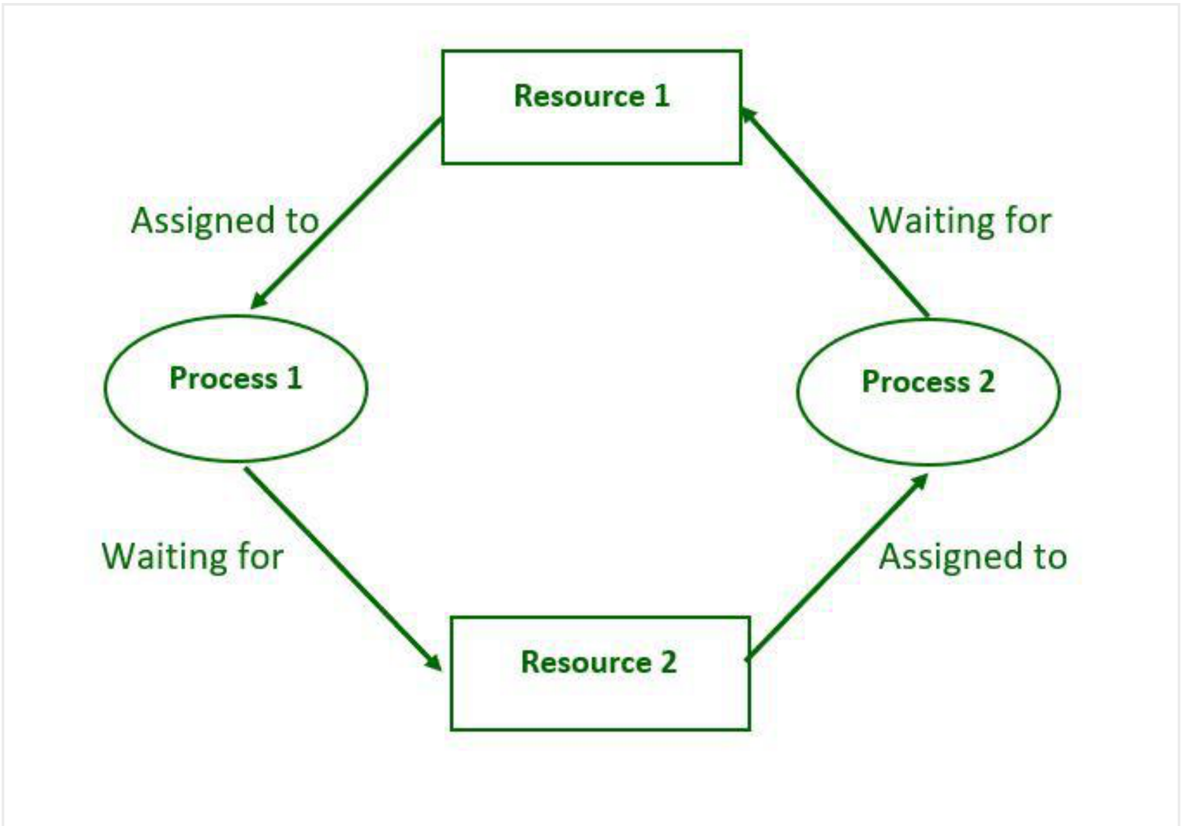


Figure: Deadlock in Operating system

Deadlock

How a Deadlock Can Occur?

Consider a simple scenario that includes two processes (Process A and Process B) and two resources (Resource 1 and Resource 2). Let's see that both processes begin execution at the same time.

- Process 1 obtains Resource 1.
- Process 2 obtains Resource 2.

We are currently in the following situation:

- Process 1 possesses Resource 1.
- Process 2 possesses Resource 2.

Let us now see what happens next: Process 1 requires Resource 2 to continue execution but is unable to do so because Process 2 is currently holding Resource 2. Similarly, Process 2 requires Resource 1 to continue execution but is unable to do so because Process 1 is currently holding Resource 1. Both processes are now stuck in a loop:

- Process 1 is awaiting Resource 2 from Process 2.
- Process 2 is awaiting Resource 1 from Process 1.

We have a deadlock because neither process can release the resource it is holding until it completes its task, and neither can proceed without the resource the other process is holding. Both processes are effectively "deadlocked," unable to move forward. To break the deadlock and free up resources for other processes in this situation, an external intervention, such as the operating system killing one or both processes, would be required. Deadlocks are undesirable in operating systems because they waste resources and have a negative impact on overall system performance and responsiveness. To prevent deadlocks, various resource allocation and process scheduling algorithms, such as deadlock detection and avoidance, are employed.

Necessary Conditions for the Occurrence of a Deadlock

Let's explain all four conditions related to deadlock in the context of the scenario with two processes and two resources:

- Mutual Exclusion
- Hold and Wait
- No Pre Emption
- Circular Wait

1. Mutual Exclusion

Mutual Exclusion condition requires that at least one resource be held in a non-shareable mode, which means that only one process can use the resource at any given time. Both Resource 1 and Resource 2 are non-shareable in our scenario, and only one process can have exclusive access to each resource at any given time. As an example:

- Process 1 obtains Resource 1.
- Process 2 acquires Resource 2.

2. Hold and Wait

The hold and wait condition specifies that a process must be holding at least one resource while waiting for other processes to release resources that are currently held by other processes. In our example,

- Process 1 has Resource 1 and is awaiting Resource 2.
- Process 2 currently has Resource 2 and is awaiting Resource 1.
- Both processes hold one resource while waiting for the other, satisfying the hold and wait condition.

3. No Preemption

Preemption is the act of taking a resource from a process before it has finished its task. According to the no preemption condition, resources cannot be taken forcibly from a process a process can only release resources voluntarily after completing its task.

For example – Process p1 have resource r1 and requesting for r2 that is hold by process p2. then process p1 can not preempt resource r2 until process p2 can finish his execution. After some time it try to restart by requesting both r1 and r2 resources.

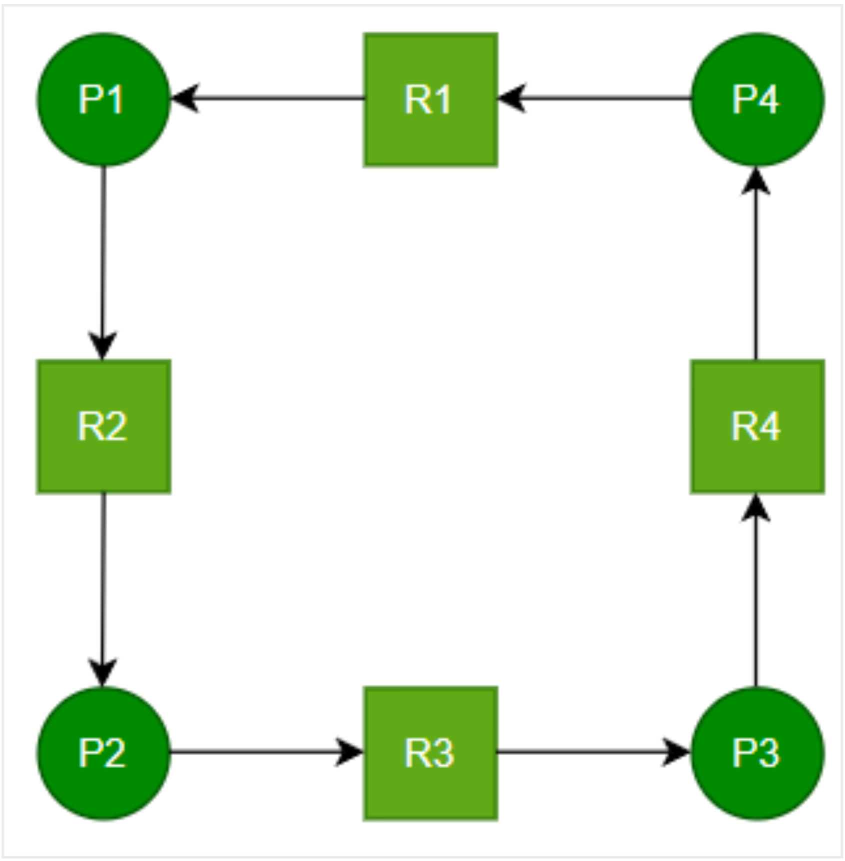
Problem – This can cause the Live Lock Problem .

What is Live Lock?
Live lock is the situation where two or more processes continuously changing their state in response to each other without making any real progress. Example:

- Suppose there are two processes 1 and 2 and two resources r1 and r2.
- Now, p1 acquired r1 and need r2 & p2 acquired r2 and need r1.
- so according to above method- Both p1 and p2 detect that they can't acquire second resource, so they release resource that they are holding and then try again.
- continuous cycle- p1 again acquired r1 and requesting to r2 p2 again acquired r2 and requesting to r1 so there is no overall progress still process are changing there state as they preempt resources and then again holding them. This the situation of Live Lock.

4. Circular Wait
Circular wait is a condition in which a set of processes are waiting for resources in such a way that there is a circular chain, with each process in the chain holding a resource that the next process needs. This is one of the necessary conditions for a **deadlock** to occur in a system.
Example: Imagine four processes—**P1**, **P2**, **P3**, and **P4**—and four resources—**R1**, **R2**, **R3**, and **R4**.

- **P1** is holding **R1** and waiting for **R2** (which is held by P2).
- **P2** is holding **R2** and waiting for **R3** (which is held by P3).
- **P3** is holding **R3** and waiting for **R4** (which is held by P4).
- **P4** is holding **R4** and waiting for **R1** (which is held by P1).



Circular wait example

This forms a circular chain where every process is waiting for a resource held by another, creating a situation where no process can proceed, leading to a deadlock.

Conclusion

Deadlocks are major problems in computers in which two or more processes remain blocked forever, each waiting for the other to release resources. A deadlock requires four conditions: mutual exclusion, hold and wait, no preemption, and circular wait. Deadlocks consume resources and reduce system performance, hence their avoidance, detection, and resolution are critical in operating systems. Deadlocks are managed and mitigated using a variety of algorithms and tactics, which ensure system stability and efficiency.

Handling Deadlocks

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.

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:

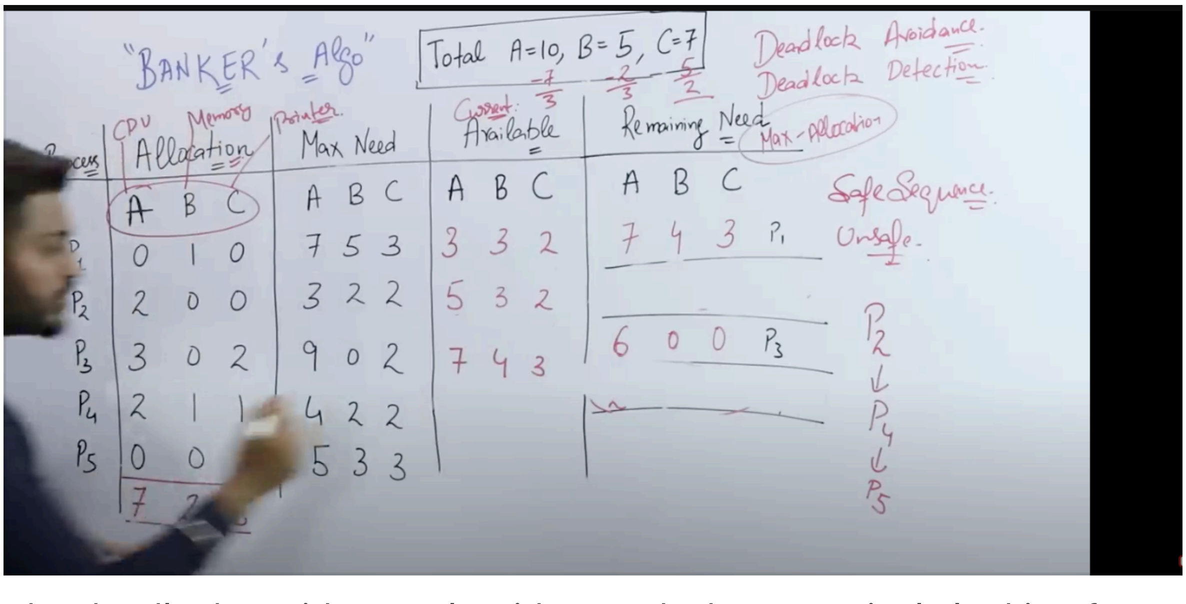
- long waiting time required
- inefficient use of allocated resource
- A process may not know all the required resources in advance

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 (Bankers Algorithm)



"BANKER's Algo"

Total A=10, B=5, C=7

Deadlock Avoidance.
Deadlock Detection.

Process	Allocation (CPU, Memory, Printer)			Max Need			Current Available			Remaining Need = Max - Allocation			
	A	B	C	A	B	C	A	B	C	A	B	C	
P1	0	1	0	7	5	3	3	3	2	7	4	3	P1
P2	2	0	0	3	2	2	5	3	2				
P3	3	0	2	9	0	2	7	4	3	6	0	0	P3
P4	2	1	1	4	2	2							
P5	0	0	[illegible]	5	3	3							
	7	2	[illegible]										

Safe Sequence.
Unsafe.

P2 ↓ P4 ↓ P5

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

3. Deadlock Detection & Recovery

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.

Conclusion

Deadlock represents a significant challenge in the realm of operating systems and resource management. It arises when processes become indefinitely blocked, each waiting for resources held by others, leading to a complete halt in system operations. By recognizing the four necessary conditions for deadlock—mutual exclusion, hold and wait, no preemption, and circular wait—developers and system administrators can implement effective strategies to prevent or resolve such situations.