Deadlock definition

Deadlock is a situation in an operating system where multiple processes get stuck because each is waiting for a resource that another process holds. Since none of them can proceed, the system stops making progress.

1. Understanding Deadlock

Imagine a traffic jam at an intersection where cars block each other:

- Each car wants to move forward but another car is blocking the way.
- Since no car can move, they remain stuck, causing deadlock.

In an **Operating System (OS)**, this happens when processes hold resources and wait for more, leading to a similar blockage.

2. Conditions Required for Deadlock

A deadlock can only occur when all **four** conditions are present at the same time:

✅ 1. Mutual Exclusion

- Resources cannot be shared; only one process can use a resource at a time.
- Example: If a printer is being used by one process, another process cannot use it until it is released.

✅ 2. Hold and Wait

- A process holding a resource must wait for additional resources held by other processes.
- Example: Process A is using the printer but also needs a scanner. The scanner is used by Process B, who also needs the printer.

✅ 3. No Preemption

- Resources cannot be taken away; a process must voluntarily release them.

- Example: The OS cannot force Process B to give up the scanner, even though Process A needs it.

✅ 4. Circular Wait

- A set of processes form a circular chain where each is waiting for a resource held by the next process in the chain.
- Example:
- Process A → Needs Resource B (held by Process B)
- Process B → Needs Resource C (held by Process C)
- Process C → Needs Resource A (held by Process A)

  This creates a **loop**, causing deadlock.

If all these conditions occur **together**, deadlock happens.

3. Examples of Deadlock

Example 1: Four Processes and Four Resources

Imagine four processes (P1, P2, P3, P4) and four resources (R1, R2, R3, R4):

- P1 holds R1 and waits for R2.
- P2 holds R2 and waits for R3.
- P3 holds R3 and waits for R4.
- P4 holds R4 and waits for R1.

Since each process is stuck waiting for a resource that another process holds, no process can proceed, leading to deadlock.

Example 2: Traffic Deadlock

- Four cars enter an intersection, each blocking the other.
- No car can move forward, leading to a permanent standstill.
- The same principle applies in an OS when processes lock each other out.

4. Methods for Handling Deadlocks

Since deadlocks affect system performance, OS designers use different techniques to **prevent**, **detect**, or **resolve** them.

✅ Deadlock Prevention

- **Break at least one of the four conditions** so deadlock never happens.
- **Methods to prevent deadlock:**
    1. **No Circular Wait:** Impose an ordering for resource requests.
    2. **No Hold and Wait:** Require processes to request all resources at once.
    3. **Allow Preemption:** Permit the system to take back resources forcibly.

✅ Deadlock Avoidance

- The system carefully checks before granting resources to avoid deadlock.
- The **Banker's Algorithm** helps decide if granting resources is safe.

✅ Deadlock Detection

- The OS monitors processes and **Resource Allocation Graphs** to check for deadlocks.
- If deadlock is found, the system **takes action**.

✅ Deadlock Recovery

Once a deadlock is detected, the OS must resolve it:

1. **Terminate a process** to free resources.
2. **Force a process to release resources** and restart it later.
3. **Restart the system** (last resort).

5. Real-World Applications & Importance

Deadlock management is crucial for:

- **Multitasking systems:** Preventing conflicts when multiple applications run simultaneously.
- **Databases:** Avoiding transaction locks in banking and online services.

- **Cloud computing:** Managing shared resources effectively.
- **Operating systems:** Ensuring stability in Windows, Linux, and macOS.

6. Summary Table

Deadlock Strategy	How It Works	Example
Prevention	Stops deadlocks before they happen	Avoid circular wait
Avoidance	Ensures safe resource allocation	Banker's Algorithm
Detection	Identifies deadlocks in running systems	Resource Allocation Graph
Recovery	Resolves a detected deadlock	Terminate or restart processes

Deadlocks can cause serious system slowdowns, crashes, and failures. Proper handling ensures smooth and efficient computing.