Deadlock Detection and Prevention

Deadlock occurs when multiple processes in an operating system are stuck because each is waiting for a resource held by another process. To manage deadlocks, operating systems use **deadlock detection** (identifying deadlocks after they occur) and **deadlock prevention** (stopping them before they happen).

1. Deadlock Detection

Deadlock detection involves identifying a deadlock in the system after it has occurred. The operating system checks for cycles in resource allocation to see if processes are stuck.

Methods of Deadlock Detection

1. **Resource Allocation Graph (RAG)** 📊
    - The OS maintains a graph showing which processes are holding or requesting resources.
    - If a **cycle** exists in the graph, deadlock has occurred.
2. **Deadlock Detection Algorithm**
    - Similar to Banker's Algorithm, the system analyzes resource requests and allocations.
    - If a process cannot proceed, the OS marks it as **deadlocked**.

How Deadlock Detection Works

1. The operating system maintains a **Resource Allocation Graph (RAG)**.
2. It monitors which processes are holding and requesting resources.
3. The system **checks for cycles** in the graph to determine if a deadlock exists.
4. If a deadlock is detected, **deadlock recovery methods** are used to resolve it.

Deadlock Detection Algorithm

The system keeps track of:

- **Available resources** – Free resources in the system.

- **Allocated resources** – Resources already assigned to processes.
- **Resource requests** – Requests made by processes.

The detection algorithm scans the system to find:

- Any **process stuck** without progressing.
- A **cycle in resource dependency**, confirming deadlock.

Example of Deadlock Detection

Consider four processes **P1, P2, P3, and P4**, each needing a resource.

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

This forms a **cycle**, meaning a deadlock has occurred.

Deadlock Recovery Methods

Once detected, the system must resolve the deadlock:

- **Terminate one or more processes** to free up resources.
- **Force preemption** (take resources from a process and assign them elsewhere).
- **Restart the system** (last resort).

Deadlock detection is useful in **dynamic systems** but requires frequent monitoring, which can slow performance.

2. Deadlock Prevention

Deadlock prevention **stops deadlocks before they happen** by ensuring that at least one of the four necessary conditions for deadlock does not occur.

Methods of Deadlock Prevention

To prevent deadlock, the OS eliminates one or more of the **four conditions** that cause it:

1. **Eliminate Mutual Exclusion** 🚫🔒
   - Allow multiple processes to share resources instead of enforcing exclusive access.
   - Example: Instead of locking files, make some files **read-only** so multiple processes can access them.
2. **Eliminate Hold and Wait** 🚫⏳
   - Require processes to request all required resources upfront instead of holding some and waiting for others.
   - Example: A process must request **printer + scanner together**, not one at a time.
3. **Allow Preemption** 🔄🔄
   - The system can **take back** a resource from one process and assign it to another.
   - Example: If a process is holding a printer for too long, the OS can reassign it to another process.
4. **Avoid Circular Wait** 🔄✂
   - Impose a strict **ordering** on resource requests so processes request resources in a specific order.
   - Example: Assign priorities to resources like:
     - First request **CPU** → then **Memory** → then **Printer**.

By eliminating at least one of these conditions, deadlocks **cannot occur**.

3. Comparison: Detection vs. Prevention

Method	Purpose	How It Works	Use Case
Deadlock Detection	Finds deadlocks	Checks resource usage for cycles	Used when prevention is difficult
Deadlock Prevention	Avoids deadlocks	Removes one of the four deadlock conditions	Used when planning resource allocation

Key Differences

- **Detection** identifies deadlocks **after** they occur.
- **Prevention** ensures deadlocks **never happen**.

Deadlock **prevention is preferable** as it avoids performance issues, but **detection is necessary** in cases where prevention is impractical.

4. Real-World Applications of Deadlock Management

Deadlock prevention and detection are crucial in:

- **Multitasking OS:** Ensuring multiple applications run smoothly.
- **Databases:** Avoiding transaction locks in banking and online services.
- **Cloud Computing:** Managing shared virtual resources.
- **Networking:** Avoiding deadlocks in data transmission.