Operating Systems

Course Code: **71203002004**

Deadlock Detection and Recovery

by -

Minal Rajwar

Deadlock Detection and Recovery

Deadlock Detection and Recovery is used in operating systems to find and solve deadlocks. A deadlock happens when two or more processes are stuck, waiting for each other's resources.

- **Detection** means checking if processes are blocked, using methods like Resource Allocation Graphs (RAG) or Wait-for Graphs.

- **Recovery** means fixing the deadlock so the system can work again. This can be done by ending a process, taking resources back, or rolling back a process.

Approaches to Deadlock Handling

Prevention – The OS avoids deadlocks by keeping the system in a safe state using resource allocation methods like the **Banker's Algorithm**.

Detection and Recovery – If deadlocks occur, the OS:

- **Detects** them using techniques like the **Wait-for Graph**.
- **Recovers** by freeing resources through methods such as **rollback** or **process termination (abort)**, so the system can continue running.

Difference between Prevention and Detection/ Recovery

- **Prevention**: Tries to stop deadlocks before they happen by controlling how resources are given to processes.

- **Detection & Recovery**: Deals with deadlocks after they occur by first finding them, then fixing them using methods like rollback or process termination.

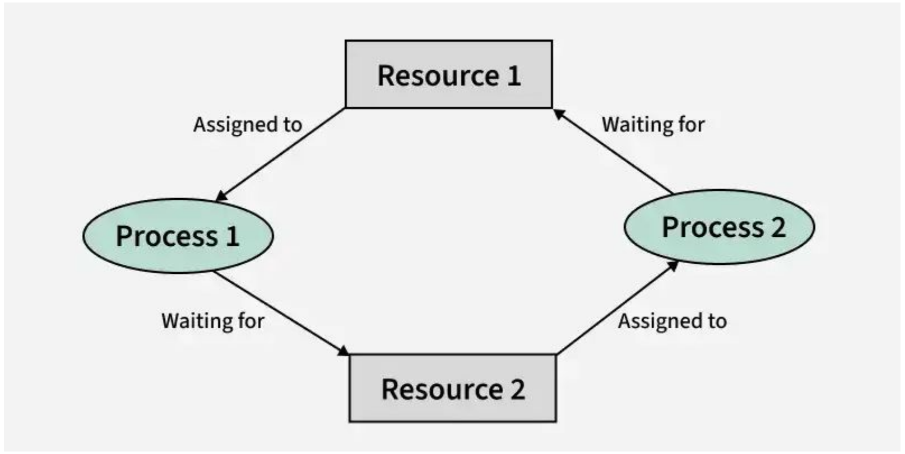
Deadlock handling is important for system stability and performance. The choice between prevention or detection/recovery depends on the system’s needs, performance goals, and complexity.

Deadlock Detection (Single Instance of Resources)

When each resource has only **one instance**, deadlock can be detected by checking for a **cycle** in the **Resource Allocation Graph (RAG)**.

If a cycle exists, it means a deadlock has definitely occurred.

Example: If R1 → P1 → R2 → P2 → R1 forms a cycle, then **deadlock is confirmed**.



Example:

Process	Holds R1	Holds R2	Requests R1	Requests R2
P1	1	0	0	1
P2	0	1	1	0

Explanation:

P1 is holding **R1** and wants **R2**.

P2 is holding **R2** and wants **R1**.

RAG:

From	To
P1	R2
P2	R1
R1	P1
R2	P2

Observation

- If you follow the arrows: **P1 → R2 → P2 → R1 → P1**, there is a **cycle**.
- A cycle in the RAG indicates a **potential deadlock**.

Conclusion:

- Since both processes are waiting for each other's resources, **deadlock occurs**.
- To resolve it, we can use **recovery methods** like killing a process, rolling back, or preempting resources.

Difference between Prevention and Detection/ Recovery…

2. **Multiple Instances of Resources**

- If resources have more than one instance, just finding a cycle is **not enough** to confirm deadlock.
- The system may or may not be in deadlock depending on the situation.
- Algorithms like a modified **Banker's Algorithm** are used to check for deadlocks.

3. **Wait-For Graph Algorithm**

- Used when resources have multiple instances.
- A **Wait-For Graph** shows which process is waiting for which other process.
- If a cycle exists in this graph, a **deadlock is detected**.

Banker’s Algorithm

- Used for **deadlock detection and prevention** in operating systems.
- Works like a banker lending money: only grants resources if the system remains in a **safe state**.

Key Data Structures:

1. **Available** – Vector of length m showing how many resources of each type are free.
2. **Allocation** – Matrix $n \times m$ showing resources currently allocated to each process.
3. **Request** – Matrix $n \times m$ showing the additional resources each process needs.

Example:

Process	Allocation (A B C)	Request (A B C)
P0	0 1 0	0 0 0
P1	2 0 0	2 0 2
P2	3 0 3	0 0 0
P3	2 1 1	1 0 0
P4	0 0 2	0 0 2

Available Resources: [0, 0, 0]

Step	Work (A B C)	Finish Vector	Action
1	0 0 0	F F F F F	Select P0
2	0 1 0	T F F F F	Select P2
3	3 1 3	T F T F F	Select P1
4	5 1 3	T T T F F	Select P3
5	7 2 4	T T T T F	Select P4
6	7 2 6	T T T T T	All processes finished → **No Deadlock**

Advantages

- Prevents deadlocks by keeping the system in a safe state.
- Can handle multiple resource types.
- Provides efficient resource allocation.

Disadvantages

- Not practical for large systems with many processes/resources.
- Requires resource needs to be known in advance.
- May waste resources if reserved but unused.

Algorithm Steps

1. **Initialize**:
   - Work = Available (resources available to allocate)
   - Finish[i] = true if Allocation for Pi = 0, else false
2. **Find a process Pi** such that:
   - Finish[i] == false
   - Request[i] <= Work
3. **If such Pi exists**:
   - Work = Work + Allocation[i]
   - Finish[i] = true
   - Repeat Step 2
4. **If no such Pi exists**:
   - If Finish[i] == false for any i → **deadlock detected**
   - If all Finish[i] == true → **no deadlock**

Deadlock Recovery

Normal OS like Windows usually don’t use recovery (too costly), but **real-time OS** may use it.

Methods of Recovery

Killing Processes

- End all processes in deadlock at once, or kill them one by one.
- After each kill, check if deadlock is gone.
- This breaks the circular wait.

Process Rollback

- Roll back processes to a safe state saved earlier (using **checkpointing**).

Deadlock Recovery

Resource Preemption

- Take resources back from deadlocked processes and give them to others.
- Risk: may cause **starvation**.

Concurrency Control

- Prevents data conflicts in concurrent systems.
- Controls access to shared resources so processes don't block each other and cause deadlocks.

Advantages of Deadlock Detection and Recovery

Improved Stability – Prevents the system from freezing by resolving deadlocks.

Better Resource Use – Ensures resources are not wasted and remain available for processes.

Improved System Design – Gives insights into process–resource interactions, helping in system improvements.

Disadvantages of Deadlock Detection and Recovery

Performance Overhead – Regular checking slows system performance.

Complexity – Algorithms can be difficult to implement.

False Results – May wrongly detect deadlocks or miss them.

Risk of Data Loss – Rollback during recovery can cause data loss or corruption.

DISCUSSION & REVISION

1. What occurs when two or more processes wait for each other's resources?
2. Which algorithm is used to prevent deadlocks by keeping the system in a safe state?
3. What type of graph is used to detect cycles in resource allocation?
4. Which recovery method reverse a process to a previously saved state?
5. What can happen if deadlock recovery preempts resources repeatedly without care?

REFERENCES

- https://www.geeksforgeeks.org/operating-systems/deadlock-detection-recovery/
- https://www.tutorialspoint.com/deadlock-detection-algorithm-in-operating-system