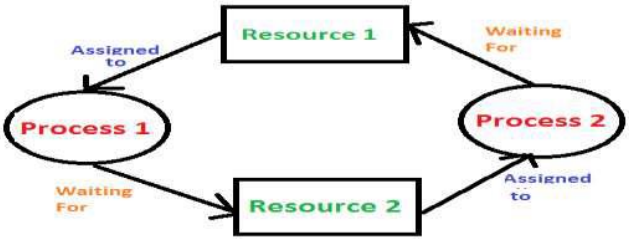
1. What is Deadlock?

- A deadlock is a situation in which a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by another process.
- Deadlock occurs in multi-processing systems where multiple processes share resources.

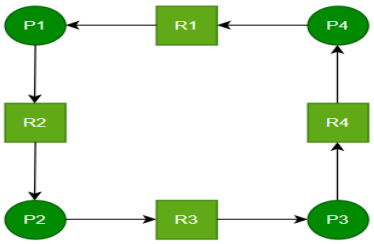


Deadlock

2. Conditions for Deadlock (Coffman's Conditions)

1. Mutual Exclusion: Only one process can use a resource at a time.
2. Hold and Wait: A process holding a resource can request additional resources.

3. No Preemption: A resource cannot be forcibly taken from a process.
4. Circular Wait: A closed chain of processes exists where each process waits for a resource held by the next process in the chain.



Circular wait

3. Deadlock Prevention & Avoidance

Deadlock Prevention

- Prevents at least one of the Coffman conditions from occurring.

- Techniques:
    - Eliminate Mutual Exclusion: Make resources shareable.
    - Eliminate Hold and Wait: Require a process to request all required resources at once.
    - Eliminate No Preemption: Allow resource preemption if needed.
    - Eliminate Circular Wait: Impose an ordering on resources.

Deadlock Avoidance

- Uses extra information about resource allocation to avoid unsafe states.
- Requires the maximum number of resources each process will request.
- Uses Banker's Algorithm to check for safe resource allocation.

4. Deadlock Detection

When deadlock avoidance is not used, deadlock detection must be performed.

Detection methods:

a. Resource Allocation Graph (RAG): If the graph contains a cycle, a deadlock exists.
b. Banker's Algorithm: Checks whether a safe sequence exists or not.
c. Wait-for Graph: A simplified RAG used when all resources are of single instances.

5. Banker's Algorithm for Deadlock Avoidance

Banker's Algorithm ensures that the system never enters an unsafe state. It uses:

a. Available Resources: Free resources in the system.
b. Max Need: Maximum resources a process may need.
c. Allocation: Resources currently allocated to each process.
d. Need: Remaining resources required (Need = Max - Allocation).

Algorithm for Deadlock Avoidance (Banker's Algorithm)

The Banker's Algorithm is used to avoid deadlocks by ensuring that the system never enters an unsafe state.

Step 1: Input Data

1. Available: Number of available instances for each resource type.
2. Max: Maximum resource demand of each process.
3. Allocation: Number of resources currently allocated to each process.
4. Need: Remaining resources required by each process (calculated as Need = Max - Allocation).

Step 2: Safe State Check (Safety Algorithm)

1. Initialize:
    - Work = Available (copy of available resources).
    - Finish[] = {false} for all processes (indicating none are finished).
    - SafeSequence[] (to store the safe execution order).

2. Find a process P[i] such that:
    - Finish[i] == false
    - Need[i] ≤ Work (for all resource types)
3. If such a process is found:
    - Allocate resources temporarily: Work = Work + Allocation[i]
    - Mark Finish[i] = true
    - Add P[i] to SafeSequence
    - Repeat for all processes.
4. If all processes finish (Finish[] = true for all), Safe State exists → Allow resource request.
5. If no process can execute, the system is in an Unsafe State → Deny request to avoid deadlock.

Algorithm for Deadlock Detection (Banker's Algorithm)

Deadlock detection is used when resource allocation is dynamic, and deadlock avoidance is not enforced.

Step 1: Input Data

1. Available: Number of free instances for each resource.
2. Allocation: Number of resources allocated to each process.
3. Request: Current resource request of each process.
4. Need: Resources still needed (Need = Max - Allocation).

Step 2: Detection Algorithm

1. Initialize:
    - Work = Available
    - Finish[] = {false} for all processes
2. Find a process P[i] such that:
    - Finish[i] == false
    - Request[i] ≤ Work (for all resource types)
3. If such a process is found:
    - Allocate resources temporarily: Work = Work + Allocation[i]
    - Mark Finish[i] = true
    - Repeat for all processes.
4. If all processes finish (Finish[] = true for all), No Deadlock.
5. If some processes are still Finish[i] == false, these processes are deadlocked.

Differences

Feature	Deadlock Avoidance (Banker's Algorithm)	Deadlock Detection
When Used	Before resource allocation	After resource allocation
Purpose	Prevents system from entering unsafe state	Detects deadlock if it occurs
Safe Sequence	Always ensures a safe sequence	No guarantee of safe sequence
Deadlock Occurrence	Never occurs if followed correctly	May occur and needs resolution