Assignment 3

**1. Define deadlock. List out the conditions that can result in resource deadlock. Explain different deadlock handling methods in detail.**

Ans:It is a situation in computing where two or more processes are unable to proceed because each is waiting for a resource that the other holds. This results in a standstill where none of the processes can continue, leading to a complete halt in the system's execution.

### Conditions for Deadlock

A deadlock situation can occur if all of the following four conditions are met simultaneously:

1. \*\*Mutual Exclusion\*\*: At least one resource must be held in a non-shareable mode, meaning that only one process can use the resource at any given time. If another process requests it, the requesting process must wait until the resource is released.

2. \*\*Hold and Wait\*\*: A process holding one or more resources is allowed to request additional resources without releasing the currently held resources. This can lead to situations where a process is holding resources and waiting for others, which contributes to the potential for deadlock.

3. \*\*No Preemption\*\*: Resources cannot be forcibly taken away from a process; they can only be released voluntarily by the process holding them. If a process needs a resource that is currently held by another process, it must wait until the resource is released.

4. \*\*Circular Wait\*\*: There must be a circular chain of processes, where each process is waiting for a resource that the next process in the chain holds. For example, Process A is waiting for a resource held by Process B, which is waiting for a resource held by Process C, and Process C is waiting for a resource held by Process A.

### Deadlock Handling Methods

There are several strategies to handle deadlocks:

1. \*\*Deadlock Prevention\*\*: This involves designing the system in such a way that one or more of the necessary conditions for deadlock are prevented from occurring. Strategies include:

- \*\*Eliminating Mutual Exclusion\*\*: This is not always practical because some resources inherently require mutual exclusion.

- \*\*Eliminating Hold and Wait\*\*: Require processes to request all the resources they need at once. If they cannot obtain all required resources, they must release any resources they hold and try again later.

- \*\*Eliminating No Preemption\*\*: Allow the system to preempt resources from processes if necessary. For example, if a process cannot obtain all the resources it needs, it may release some of the resources it holds.

- \*\*Eliminating Circular Wait\*\*: Impose a total ordering on all resource types and require processes to request resources in increasing order of enumeration. This prevents circular wait conditions.

2. \*\*Deadlock Avoidance\*\*: This method ensures that the system is always in a safe state by making decisions about resource allocation based on the current state of the system. The most common approach is:

- \*\*Banker’s Algorithm\*\*: This algorithm checks whether resource allocation will leave the system in a safe state, where a sequence of processes can complete without causing a deadlock. It dynamically assesses whether granting a resource request will still allow the system to meet the needs of all processes.

3. \*\*Deadlock Detection and Recovery\*\*: In this approach, the system allows deadlock to occur but has mechanisms to detect and recover from it. This involves:

- \*\*Detection\*\*: Periodically checking for deadlocks using algorithms that track resource allocation and process states to identify cycles in the resource allocation graph.

- \*\*Recovery\*\*: Once a deadlock is detected, the system must take action to break the deadlock. Common recovery methods include:

- \*\*Process Termination\*\*: Terminating one or more of the processes involved in the deadlock. The choice of which process to terminate can be based on factors such as process priority or the amount of resources held.

- \*\*Resource Preemption\*\*: Temporarily taking resources away from some processes and allocating them to others until the deadlock is resolved. The process with preempted resources will be restarted later.

4. \*\*Deadlock Ignorance\*\*: In some systems, deadlock is simply ignored, under the assumption that it is rare or that the cost of handling it is higher than the cost of dealing with occasional deadlocks. This approach is generally not recommended for critical systems.

Each method has its own advantages and trade-offs, and the choice of method depends on factors such as system requirements, resource usage patterns, and performance considerations.

1. **A system has two process and three resources. Each process needs a maximum of two resources.Is deadlock possible? Explain.**

Ans:To determine if deadlock is possible in the system with two processes and three resources, each process needing a maximum of two resources, we can use the \*\*Banker's Algorithm\*\* approach and \*\*theoretical deadlock analysis\*\*.

### Analysis

1. \*\*Number of Processes and Resources\*\*:

- Processes (P1 and P2): 2

- Resources: 3

- Maximum resources needed by each process: 2

2. \*\*Deadlock Conditions\*\*:

Let's check if deadlock conditions can be met.

- \*\*Mutual Exclusion\*\*: This condition is inherent and must be met for resource sharing.

- \*\*Hold and Wait\*\*: Processes may hold resources while waiting for additional resources.

- \*\*No Preemption\*\*: Resources cannot be taken away from processes forcibly.

- \*\*Circular Wait\*\*: A circular chain of processes, each waiting for a resource held by the next process, must exist.

3. \*\*Resource Allocation and Needs\*\*:

To assess deadlock possibility, consider the worst-case scenario:

- \*\*Scenario for Deadlock\*\*:

- Both processes could potentially hold 1 resource each and request the remaining resources. If each process holds 1 resource and requests the remaining 1 resource, it could lead to a circular wait situation.

Let's assume:

- Process P1 holds 1 resource and requests 1 more.

- Process P2 holds 1 resource and requests 1 more.

In this case:

- P1 cannot proceed because it needs 1 more resource which P2 might be holding.

- P2 cannot proceed because it needs 1 more resource which P1 might be holding.

\*\*The Circular Wait Condition\*\* is evident if both processes are in such a state.

### Deadlock Possibility

Using the \*\*Banker's Algorithm\*\* approach:

- The system can have a maximum of 4 resources (total available resources plus maximum request) if it is to avoid deadlock.

- \*\*Total Resources\*\*: 3

- \*\*Maximum Request by Both Processes\*\*: 2 + 2 = 4

The number of resources (3) is less than the maximum possible (4). Hence, there can be a state where all resources are allocated and both processes are waiting for additional resources.

### Conclusion

Deadlock is possible in this scenario.

- With 3 resources and the possibility that each process could hold up to 2 resources while requesting the remaining, the system can enter a deadlock state where both processes hold 1 resource each and wait for the other resource, leading to a circular wait situation.

This analysis assumes a worst-case scenario. In practice, the system might not always reach this state, but theoretically, it's possible.

1. **How can you represent a deadlock? Explain with suitable example.**

Ans:A **deadlock** occurs when processes are stuck waiting for each other to release resources.Example: Three processes (P1, P2, P3) each hold a resource and request another held by another process, forming a circular dependency.

**Conditions for Deadlock**:

Mutual Exclusion: Resources can’t be shared simultaneously.

Hold and Wait: Processes hold resources and wait for others.

No Preemption: Resources can’t be forcibly taken.

Circular Wait: Processes wait in a circular chain

**Deadlock Prevention:**

Avoid necessary conditions (e.g., allow resource sharing).

Use preemptive resource allocation.

Monitor resource allocation graph.

**Example**

P1 holds R2, requests R3.

P2 holds R3, requests R1.

P3 holds R1, requests R2.

Circular dependency leads to deadlock.

Remember, preventing deadlock is crucial for system stability!

1. **How can deadlock be prevented? Explain.**

**Ans:** \*\*Deadlock Prevention\*\* involves designing the system to ensure at least one of the necessary deadlock conditions cannot occur. Here’s how each condition can be prevented:

1. \*\*Mutual Exclusion\*\*

- \*\*Avoiding Mutual Exclusion\*\*: Share resources when possible, but this isn’t always practical for resources that inherently require mutual exclusion (e.g., printers).

2. \*\*Hold and Wait\*\*

- \*\*Request All Resources at Once\*\*: Require processes to request all needed resources at once. If they cannot get all resources, they release any they hold and try again later.

- \*\*Resource Allocation Ordering\*\*: Require processes to request resources in a predefined order to avoid circular wait.

1. \*\*No Preemption\*\*

- \*\*Allow Preemption\*\*: If a process is waiting for additional resources, it may have some of its current resources taken away and allocated to other processes. The process will then try again later.

1. \*\*Circular Wait\*\*

- \*\*Resource Ordering\*\*: Define a global order for resource requests. Processes must request resources in this order, preventing circular wait.

**6 Describe resource allocation graph. Explain how resource graph can be used for detecting deadlocks?**

Ans:

A \*\*Resource Allocation Graph (RAG)\*\* is a visual tool used to represent the allocation of resources in a system and to help detect potential deadlocks. It consists of nodes and edges:

- \*\*Nodes\*\*:

- \*\*Process Nodes\*\*: Represent processes in the system.

- \*\*Resource Nodes\*\*: Represent resources available in the system.

- \*\*Edges\*\*:

- \*\*Request Edge\*\*: An edge from a process node to a resource node, indicating that the process is requesting the resource.

- \*\*Allocation Edge\*\*: An edge from a resource node to a process node, indicating that the resource is currently allocated to the process.

A Resource Allocation Graph helps detect deadlocks by analyzing the structure of the graph. The key steps are:

1. \*\*Construct the Graph\*\*:

- Draw nodes for each process and each resource.

- Add request edges from processes to resources if a process is requesting a resource.

- Add allocation edges from resources to processes if a resource is currently allocated to a process.

2. \*\*Detect Cycles\*\*:

- \*\*Cycle Detection\*\*: A cycle in the graph indicates a deadlock. Specifically, a cycle means that there is a circular chain of processes and resources where each process is waiting for a resource held by the next process in the chain.

- \*\*Cycle Analysis\*\*: Identify if there are any cycles in the graph. If a cycle exists, it means that the processes in the cycle are in a state of deadlock because they are all waiting for resources that are held by each other.

3. \*\*Interpreting the Graph\*\*:

- If the graph contains no cycles, the system is in a safe state, and no deadlock is occurring.

- If the graph contains one or more cycles, deadlock is present, and the system must take action to break the deadlock.

1. **Define the term indefinite postponement. How does it differ from deadlock?**

**Ans:** \*\*Indefinite Postponement\*\*, also known as \*\*Starvation\*\*, occurs when a process is perpetually delayed from accessing resources it needs to proceed. This typically happens because other processes continuously receive the resources before the delayed process gets a chance.

Differences Between Deadlock and Indefinite Postponement

1. \*\*Definition\*\*:

- \*\*Deadlock\*\*: A situation where a set of processes are stuck in a cycle of waiting for each other to release resources, causing all of them to be unable to proceed. Each process in the cycle holds resources that other processes need, leading to a standstill.

- \*\*Indefinite Postponement\*\*: A scenario where a process is never given the opportunity to execute because other processes keep getting the resources it needs, leading to an indefinite delay for the affected process.

2. \*\*System State\*\*:

- \*\*Deadlock\*\*: All processes involved in the deadlock are in a state where they are waiting indefinitely for resources held by each other. The system as a whole is stuck.

- \*\*Indefinite Postponement\*\*: Only the affected process or processes are delayed. The system as a whole may still be functional, but some processes are unfairly deprived of resources.

3. \*\*Resolution\*\*:

- \*\*Deadlock\*\*: Requires intervention such as deadlock detection and recovery or prevention mechanisms to break the cycle and restore normal operation.

- \*\*Indefinite Postponement\*\*: Often resolved through fair resource allocation policies or scheduling algorithms that ensure all processes eventually get a chance to execute (e.g., using aging, priority adjustments, or round-robin scheduling).

4. \*\*Impact on Processes\*\*:

- \*\*Deadlock\*\*: All processes in the deadlock are stalled and unable to proceed. No progress is made for any of the involved processes.

- \*\*Indefinite Postponement\*\*: Only specific processes are affected by being delayed, while other processes may continue to progress normally.

In summary, while both deadlock and indefinite postponement involve processes being unable to proceed due to resource issues, deadlock results in a complete standstill of all involved processes, whereas indefinite postponement involves unfair delays for certain processes while others may continue normally.