Deadlock overview

Deadlock is a situation in a computer system where two or more processes are unable to proceed because each process is waiting for another to release a resource or a lock. In other words, a deadlock occurs when two or more processes are stuck in a loop where each process is waiting for the other to finish, and none of the processes can continue.

Deadlock can occur in any multi-process system, including operating systems, database systems, and distributed systems. Deadlock can cause significant problems, including system crashes, lost data, and reduced system performance.

There are four necessary conditions for a deadlock to occur: mutual exclusion, hold and wait, no preemption, and circular wait. These conditions must all be present for a deadlock to occur. To prevent or resolve deadlocks, various techniques can be used, including resource allocation graphs, deadlock detection algorithms, and deadlock prevention methods.

System Model in deadlock

In order to understand deadlocks, it is important to have a system model in place. A system model is a way to represent the various resources and processes within a system. Here are some components of a system model that are relevant to deadlocks:

Resources: These are entities that are needed by processes to complete their tasks. Resources can be divided into two types: preemptable resources and non-preemptable resources. Preemptable resources can be taken away from a process while it is still being used. Non-preemptable resources cannot be taken away from a process until it has completed its task.

Processes: These are entities that are executing within the system. Each process requires one or more resources to complete its task.

Request: A process can request a resource by sending a request to the resource manager. The request specifies which resource the process wants to use.

Allocation: The resource manager can allocate resources to processes. Once a resource is allocated, it cannot be allocated to another process until the first process has released it.

Hold and wait: A process may hold a resource while it is waiting for another resource to become available.

Circular wait: A set of processes is deadlocked if each process in the set is waiting for a resource that can only be released by another process in the set.

By modeling these components and understanding the necessary conditions for a deadlock to occur (mutual exclusion, hold and wait, no preemption, and circular wait), we can design systems and develop algorithms to prevent and resolve deadlocks.

Deadlock Characterization

Deadlock can be characterized by the following four conditions, which must all be present simultaneously for a deadlock to occur:

Mutual exclusion: At least one resource must be held in a non-sharable mode, meaning that only one process at a time can use the resource.

Hold and wait: A process holding at least one resource is waiting to acquire additional resources held by other processes. In other words, a process has resources and is waiting for more.

No preemption: Resources cannot be forcibly taken away from a process holding them. The only way a process can release a resource is by voluntarily releasing it after completing its task.

Circular wait: A set of two or more processes are blocked and waiting for resources held by each other. In other words, there is a cycle of processes and resources, where each process is waiting for a resource that is held by another process in the cycle.

If these four conditions are met, a deadlock will occur, and the affected processes will be stuck in a loop where each process is waiting for the other process to release its resources. Deadlocks can cause significant problems, including system crashes, lost data, and reduced system performance. It is important to have strategies and techniques in place to prevent and resolve deadlocks.

Methods for Handling Deadlocks

There are several methods for handling deadlocks, including prevention, avoidance, detection, and recovery.

Deadlock Prevention: The goal of deadlock prevention is to ensure that the system will never enter a deadlock state. This can be achieved by eliminating one or more of the four necessary conditions for deadlock: mutual exclusion, hold and wait, no preemption, and circular wait. Some techniques for preventing deadlocks include using a single instance of a resource, requiring processes to request and obtain all necessary resources before execution, and using a timeout mechanism to prevent a process from waiting indefinitely.

Deadlock Avoidance: The goal of deadlock avoidance is to dynamically allocate resources to processes in a way that avoids deadlocks. This can be achieved by using an algorithm that tracks the available resources and the processes that are waiting for them, and that will only grant resource requests if it is safe to do so. One commonly used algorithm for deadlock avoidance is the banker's algorithm.

Deadlock Detection: The goal of deadlock detection is to detect when a deadlock has occurred and take steps to recover from it. This can be achieved by periodically examining the state of the system to identify the presence of a deadlock. Once a deadlock is detected, processes can be terminated or resources can be forcibly removed to break the deadlock.

Deadlock Recovery: The goal of deadlock recovery is to recover from a deadlock after it has occurred. This can be achieved by terminating one or more processes to release resources, rolling back transactions, or initiating a system-wide restart.

Each method has its own advantages and disadvantages, and the choice of method will depend on the specifics of the system and the requirements of the application. A combination of these methods may also be used to provide comprehensive deadlock handling in a system.

Deadlock Prevention

Deadlock prevention involves designing a system in such a way that it is impossible for a deadlock to occur. Prevention techniques involve eliminating one or more of the necessary conditions for deadlocks: mutual exclusion, hold and wait, no preemption, and circular wait. Here are some techniques for preventing deadlocks:

Mutual Exclusion Avoidance: Resources that can be safely shared between processes should not be made exclusive. This can be achieved by making the resource sharable or by introducing a new resource that can be shared by multiple processes.

Hold and Wait Avoidance: In this technique, a process should request all its required resources at once before it starts its execution. This means that a process cannot request for a new resource while holding onto another resource.

No Preemption: A process should release all resources before it can request new resources. If a process requires a new resource, it must release all its current resources and then request the new ones. If a process cannot release its current resources, then the system can forcefully release them.

Circular Wait Avoidance: Assign a numerical value to each resource and require that processes request resources in a strict order of increasing value. This means that a process can only request a resource that has a lower value than any resource that it is currently holding.

Deadlock prevention techniques may result in resource wastage, as resources may be reserved but never used. It is important to strike a balance between resource utilization and deadlock prevention. Additionally, not all systems can be designed to prevent deadlocks; some systems require dynamic resource allocation, which makes deadlock prevention difficult or impossible. In such cases, avoidance, detection, and recovery techniques may be more appropriate.

Deadlock Avoidance

Deadlock avoidance is a technique used to dynamically allocate resources to processes in such a way that the system can avoid deadlock. This technique involves checking whether a resource allocation request by a process will lead to a deadlock, and then deciding whether to grant the request or not.

One common algorithm for deadlock avoidance is the banker's algorithm, which works as follows:

- Determine the maximum demand of each process. This is the total number of resources that a process may need to complete its execution.
- Determine the available resources in the system. This is the number of resources that are currently not in use by any process.
- Determine the resource allocation matrix. This matrix represents the current allocation of resources to each process.
- Determine the resource need matrix. This matrix represents the number of additional resources that each process needs to complete its execution.
- Use the banker's algorithm to check if granting a resource request to a process will lead to a deadlock. If the request does not lead to a deadlock, grant the request; otherwise, deny the request and the process will have to wait.

The banker's algorithm uses a safety algorithm to determine whether granting a resource request will lead to a deadlock. The safety algorithm works by simulating the allocation of resources to each process and checking if the system can still avoid deadlock. If the system can avoid deadlock, the resource request is granted; otherwise, the request is denied.

Deadlock avoidance can be effective in preventing deadlocks, but it requires a lot of computation and may lead to low resource utilization. It is important to strike a balance between resource utilization and deadlock avoidance. Additionally, the system must be designed in such a way that the maximum demand of each process is known in advance, and the algorithm must be able to predict future resource requests accurately.

Safe State

In operating system theory, a safe state is a state in which the system can allocate resources to each process in a way that avoids a deadlock. This means that the system can allocate all the resources that each process needs to complete its execution without leading to a deadlock.

A state is considered safe if there is at least one sequence of resource allocations that can be made to the processes that will not result in a deadlock. This means that the system can allocate resources to each process without any process having to wait indefinitely for a resource that is held by another process.

The safe state is determined using the banker's algorithm, which checks whether a given state is safe or not. The banker's algorithm uses a safety algorithm to determine whether a state is safe. The safety algorithm works by simulating the allocation of resources to each process and checking if the system can still avoid deadlock. If the system can avoid deadlock, the state is considered safe; otherwise, it is not safe.

The concept of a safe state is important in the context of resource allocation in operating systems. It helps the system to determine if it is safe to allocate resources to a process without causing a deadlock. If the state is not safe, the system must take steps to avoid or recover from a deadlock.

Resource-Allocation Graph Algorithm in points

The resource-allocation graph algorithm is a technique used to detect deadlocks in a system. The algorithm works by analyzing the resource allocation graph, which is a directed graph that represents the allocation of resources to processes. The following are the steps involved in the resource-allocation graph algorithm:

- Construct the resource allocation graph. The graph has two types of nodes: process nodes and resource nodes. Each process node represents a process, and each resource node represents a resource.

- Draw an edge from a process node to a resource node if the process is currently holding the resource. Draw an edge from a resource node to a process node if the resource is currently allocated to the process.

- Analyze the graph to determine if a cycle exists. A cycle in the graph indicates a deadlock.

- If a cycle exists, determine the processes that are involved in the cycle. These processes are deadlocked.

- Resolve the deadlock by either aborting one or more processes or by releasing one or more resources. The goal is to break the cycle and prevent the deadlock from occurring again.

The resource-allocation graph algorithm is a simple and effective technique for detecting deadlocks. However, it may not be suitable for large and complex systems. In such cases, more advanced techniques such as the banker's algorithm may be more appropriate.

Banker's Algorithm in points

The banker's algorithm is a deadlock-avoidance algorithm used in operating systems to prevent deadlocks. It works by predicting whether a particular allocation of resources will lead to a safe state or not. The following are the steps involved in the banker's algorithm:

- Determine the maximum need of each process. This is the total number of resources that a process may need to complete its execution.
- Determine the available resources in the system. This is the number of resources that are currently not in use by any process.
- Determine the resource allocation matrix. This matrix represents the current allocation of resources to each process.
- Determine the resource need matrix. This matrix represents the number of additional resources that each process needs to complete its execution.
- Use the banker's algorithm to check if granting a resource request to a process will lead to a safe state. If the request leads to a safe state, grant the request; otherwise, deny the request and the process will have to wait.
- Use the safety algorithm to determine if the current state is safe. The safety algorithm works by simulating the allocation of resources to each process and checking if the system can still avoid deadlock.
- If the current state is safe, allocate the resources to the process. Otherwise, the process must wait until a safe state can be achieved.

The banker's algorithm is a powerful technique for preventing deadlocks. However, it has certain limitations, such as the assumption that the maximum need of each process is known in advance, and the fact that it may lead to low resource utilization. Nevertheless, it remains a valuable tool for managing the allocation of resources in operating systems.

Safety Algorithm in points

The safety algorithm is a part of the banker's algorithm, used to determine if a system is in a safe state, i.e., whether there is a sequence of resource allocation that will not lead to a deadlock. The following are the steps involved in the safety algorithm:

- Initialize two data structures: the work vector and the finish vector. The work vector represents the number of available resources of each type, and the finish vector represents whether each process can complete its execution.

- Copy the available resources of each type into the work vector.

- Initialize the finish vector to false for all processes.

- Search for a process that can finish its execution by checking if its resource need is less than or equal to the work vector. If a process is found, mark it as finished by setting its corresponding entry in the finish vector to true and add its allocated resources to the work vector.

- Repeat step 4 until no more processes can be finished.

- If all processes are marked as finished in the finish vector, then the system is in a safe state. Otherwise, the system is not in a safe state, and a deadlock may occur.

The safety algorithm is used to determine if a system is in a safe state before granting resource requests to processes. If the system is not in a safe state, then the resource request must be denied to avoid a potential deadlock. The safety algorithm ensures that a safe sequence of resource allocation can be found to avoid a deadlock.

Resource-Request Algorithm (The Bankers Algorithm) in points

The resource-request algorithm, also known as the banker's algorithm, is used to determine if a process can request a resource without leading to a deadlock. The following are the steps involved in the resource-request algorithm:

- When a process requests a resource, determine if the request can be granted by checking if the number of available resources of that type is greater than or equal to the requested amount.
- If the request can be granted, temporarily allocate the resource to the process and simulate the safety algorithm to determine if the system will be in a safe state.
- If the system will be in a safe state, grant the request and update the allocation matrix and the available resources.
- If the system will not be in a safe state, deny the request and keep the process in a blocked state until the requested resources become available.
- If the process releases a resource, update the allocation matrix and the available resources, and check if any blocked processes can be unblocked and their requests granted.

The resource-request algorithm is used to ensure that the system remains in a safe state and avoid deadlocks by checking if granting a resource request will lead to a safe state. It is a powerful technique for managing the allocation of resources in operating systems and can be used in conjunction with the safety algorithm to prevent deadlocks.

Deadlock Detection

Deadlock detection is a technique used in operating systems to identify the presence of deadlocks in a system. The following are the steps involved in deadlock detection:

- Construct a resource allocation graph or a process resource table to represent the resource allocation status of the system.

- Apply the algorithm to detect if a cycle exists in the graph or table. If a cycle exists, it indicates that a deadlock has occurred.

- Once a deadlock is detected, the system can either terminate one or more processes to break the deadlock or use other techniques, such as resource preemption, to resolve the deadlock.

- If a process is terminated, its resources are released, and the deadlock detection algorithm is applied again to see if any other deadlocks are present in the system.

Deadlock detection is used when other techniques such as prevention, avoidance, or resource preemption are not feasible or have not been implemented. It can be used in conjunction with other techniques to manage the allocation of resources and prevent deadlocks from occurring. However, it has certain limitations, such as the time required to detect deadlocks, and the fact that it cannot prevent deadlocks from occurring in the first place.

Detection-Algorithm Usage in points

The detection algorithm is used to identify the presence of deadlocks in a system. Here are some points on how the detection algorithm is used:

- A resource allocation graph or a process resource table is constructed to represent the resource allocation status of the system.
- The detection algorithm is applied to check if a cycle exists in the graph or table. If a cycle exists, it indicates that a deadlock has occurred.
- Once a deadlock is detected, the system can either terminate one or more processes to break the deadlock or use other techniques, such as resource preemption, to resolve the deadlock.
- If a process is terminated, its resources are released, and the detection algorithm is applied again to see if any other deadlocks are present in the system.
- Deadlock detection is used when other techniques such as prevention, avoidance, or resource preemption are not feasible or have not been implemented.
- Deadlock detection is a reactive approach, as it is used to identify deadlocks after they have occurred. It is not a proactive approach, as it does not prevent deadlocks from occurring in the first place.
- The detection algorithm can be used in conjunction with other techniques to manage the allocation of resources and prevent deadlocks from occurring.

The detection algorithm has certain limitations, such as the time required to detect deadlocks, and the fact that it cannot prevent deadlocks from occurring. It is important to use other techniques to prevent deadlocks whenever possible.

Recovery From Deadlock

Recovery from deadlock is the process of resolving a deadlock situation in a system. The following are some of the ways to recover from deadlock:

Process termination: In this approach, one or more processes involved in the deadlock are terminated, releasing the resources they were holding. This frees up resources that can then be used by other processes to complete their tasks. However, this approach can lead to data loss and can be disruptive to the system.

Resource preemption: In this approach, the operating system forcibly takes away some resources from one or more processes involved in the deadlock, freeing them up for other processes to use. The preempted processes are then blocked until the required resources become available again. This approach is less disruptive than process termination but can lead to resource starvation and reduced system performance.

Rollback: In this approach, the system rolls back the transactions of one or more processes involved in the deadlock to a previous consistent state. This frees up resources and allows the processes to resume their execution from the previous state. However, this approach can be time-consuming and can result in loss of data.

Killing all processes and restarting the system: In this approach, all processes involved in the deadlock are terminated, and the system is restarted. This clears all resources and allows the system to resume normal operation. However, this approach can result in significant downtime and data loss.

Recovery from deadlock is a critical process that must be carefully planned and executed to minimize disruptions and data loss. It is essential to use a combination of techniques, such as prevention, avoidance, and detection, along with recovery, to effectively manage deadlocks in a system.