**Table of contents**

**Deadlock…………………………………………………………….1**

**Necessary Conditions for Deadlock……………………………….2,3**

**Methods for Handling Deadlocks………………………………………………….4**

**Conclusion…………………………………………………………..5**

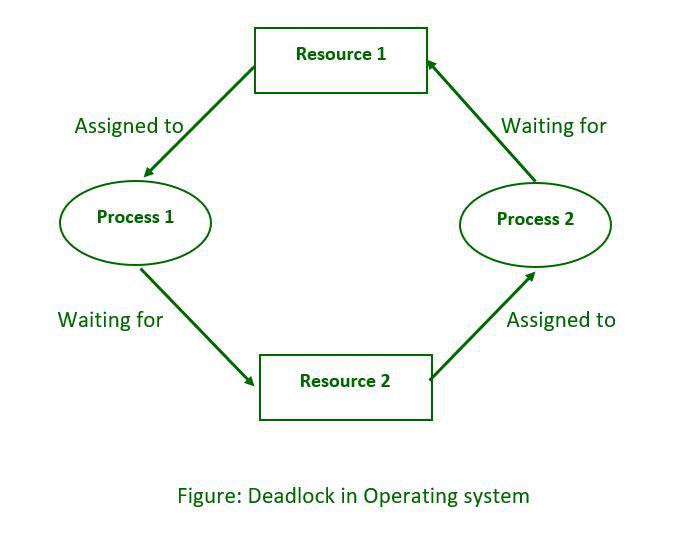
**Table of Figures**

Deadlock in operating system……………….……………………..1

1. **Deadlock:**

Deadlock is a situation where two or more processes are unable to proceed because each is waiting for the other to release a resource. It's a common limitation in mobile applications when multiple tasks are competing for limited resources. For example, if two apps are trying to access the same data simultaneously, a deadlock can occur.

To understand deadlock, let's consider a simple example. Imagine you have two apps running on your phone, App A and App B. App A needs to access a certain resource, let's say a database, to perform a task. At the same time, App B also needs to access the same database for its own task. Now, if App A locks the database and App B tries to access it, it will have to wait until App A releases the lock. Similarly, if App B locks the database and App A tries to access it, App A will have to wait. This creates a deadlock situation where both apps are waiting for each other to release the resource, but neither can proceed.

To understand deadlock, let's consider a simple example. Imagine you have two apps running on your phone, App A and App B. App A needs to access a certain resource, let's say a database, to perform a task. At the same time, App B also needs to access the same database for its own task. Now, if App A locks the database and App B tries to access it, it will have to wait until App A releases the lock. Similarly, if App B locks the database and App A tries to access it, App A will have to wait. This creates a deadlock situation where both apps are waiting for each other to release the resource, but neither can proceed.

1

1. **Necessary Conditions for Deadlock:**
2. Mutual Exclusion: This condition means that a resource can only be held by one process at a time. For example, if a process is using a printer, it has exclusive access to it, and no other process can use it simultaneously. This condition is necessary to ensure the integrity and consistency of shared resources.
3. Hold and Wait: This condition occurs when a process holds at least one resource and is waiting to acquire additional resources held by other processes. It means that a process can request resources while still holding onto the ones it already has. This can lead to a situation where multiple processes are waiting for resources to be released by other processes, resulting in a deadlock.
4. No Preemption: Preemption refers to the act of forcibly taking away a resource from a process. In the context of deadlock, this condition states that resources cannot be taken away from a process involuntarily. Resources can only be released voluntarily by the process holding them. This condition ensures that a process cannot be interrupted in the middle of its execution, which can potentially lead to inconsistencies or data corruption.
5. Circular Wait: Circular wait occurs when there is a circular chain of two or more processes, where each process is waiting for a resource held by the next process in the chain. For example, Process A is waiting for a resource held by Process B, Process B is waiting for a resource held by Process C, and so on, until Process N is waiting for a resource held by Process A. This circular dependency prevents any of the processes from progressing, resulting in a deadlock.

These four conditions together create the necessary circumstances for deadlock to occur. By understanding these conditions, system designers can implement strategies such as resource allocation algorithms, deadlock detection, and avoidance techniques to prevent or resolve deadlocks.

2

1. **Methods for Handling Deadlocks:**

**Prevention:**

To prevent deadlock, there are a few strategies that can be implemented:

* Deadlock Avoidance: This approach involves carefully analyzing resource requests and releases to ensure that granting a request will not lead to deadlock. By using algorithms like the Banker's algorithm, the system can predict and prevent potential deadlocks by granting resource requests only if it is safe to do so.
* Resource Allocation Graph: This technique uses a graphical representation of processes and resources to detect potential deadlocks. By analyzing the graph, cycles can be identified, indicating the possibility of a deadlock. To prevent deadlock, resources can be allocated in a way that avoids creating cycles in the graph.
* Deadlock Detection and Recovery: In this approach, the system periodically checks for the presence of deadlocks. If a deadlock is detected, the system can take actions such as killing or preempting processes to break the deadlock and release the resources. Alternatively, the system can roll back the progress made by processes to a safe state and restart them.
* Resource Ordering: By imposing a strict order in which resources can be requested and released, the system can prevent circular wait conditions. This can be done by assigning a numerical value or priority to each resource and ensuring that processes can only request resources in increasing order.

It's important to note that no single method can guarantee complete prevention of deadlocks. The choice of prevention strategy depends on the specific requirements and constraints of the system.

3

**Avoidance:**

To avoid deadlock, one common approach is deadlock avoidance. This involves carefully analyzing resource requests and releases to ensure that granting a request will not lead to deadlock. By using algorithms like the Banker's algorithm, the system can predict and prevent potential deadlocks by granting resource requests only if it is safe to do so. This way, the system can avoid entering into a state where deadlock is possible. It's an effective way to keep things running smoothly and prevent those frustrating deadlocks.

**Detection and Recovery:**

Another strategy to deal with deadlocks is deadlock detection and recovery. In this approach, the system periodically checks for the presence of deadlocks. If a deadlock is detected, the system can take actions such as killing or preempting processes to break the deadlock and release the resources. Alternatively, the system can roll back the progress made by processes to a safe state and restart them. It's like having a safety net in place to identify and resolve deadlocks when they occur.

**Avoidance Using Timeouts:**

Another aspect of deadlock avoidance is using timeouts. By setting timeouts on resource requests, the system can prevent processes from waiting indefinitely for a resource. If a process doesn't receive the requested resource within a specified time, it can release its held resources and try again later. This helps avoid potential deadlocks by ensuring that processes don't get stuck waiting indefinitely. It's like giving processes a time limit to acquire resources and preventing them from waiting forever.

4

1. **Conclusion:**

Deadlocks pose a significant challenge in computer systems, impacting performance and user experience. Understanding the necessary conditions and employing effective methods for prevention, avoidance, detection, and recovery is essential for maintaining system reliability and stability. By implementing appropriate strategies, system administrators and developers can minimize the occurrence and impact of deadlocks in their systems.

Reference

1. geeksforgeeks

2. Baeldung

3. Google

4. Wikipedia

5