Operating Systems Assignment

Deadlock Detection & Prevention

Submitted By: Cosc232101016 Ayesha Maryam

Cosc232101041 Aqsa Rehman

Submitted To: Mr. Muhammad Ahsan Asl

Abstract:

With the rapid advancement of technology operating systems have become more sophisticated and enhanced its efficiency. However, deadlocks have been a difficult challenge, especially in complex multitasking environments like Windows and Linux. Deadlocks in OS occurs when two or more processes are stuck, each waiting for a resource held by another process, preventing further progress. As modern OS implement deadlock detection, prevention, and recovery techniques, they still impact system performance. In this assignment we present some system models and deadlock handling techniques to deal with the problem. So, we will explore various aspects of deadlocks, including their causes, detection methods, prevention strategies, and recovery techniques. First, we discuss the fundamental conditions necessary for deadlock occurrence, Understanding these conditions is crucial for effective deadlock management. Next, it has deadlock detection methods. These methods allow the operating system to assess the system state and identify potential deadlocks. Further it includes deadlock prevention techniques, which structurally eliminate one or more of the necessary deadlock conditions.

Introduction:

Operating systems frequently experience deadlock, in which several processes are unable to continue because they are waiting on resources that are shared by one another. Developing successful deadlock management techniques requires an understanding of the four basic conditions: mutual exclusion, hold and wait, no preemption, and circular wait.

A deadlock results, for instance, when one process locks a resource while it waits for another, while the second process does the same in reverse order. Using resource allocation graphs and detection methods, deadlock detection is essential

for spotting possible system obstructions. Early detection enables prompt action to stop system crashes or slowdowns.

Deadlock prevention is an essential design objective in addition to detection. Operating systems can lessen the chance of deadlocks by structurally removing one or more deadlock circumstances. Operating systems use a variety of techniques, including detection, prevention, avoidance, and recovery, to deal with deadlocks. While prevention approaches make sure that at least one of the required deadlock criteria does not occur, detection techniques enable the system to recognize deadlocks and take corrective action.

Furthermore, the system is never put in danger thanks to deadlock avoidance strategies like dynamic resource allocation depending on process requirements. Modern operating systems minimize deadlock-related disturbances and effectively manage resources by combining these tactics.

Methodology:

Necessary conditions for deadlock:

Deadlocks arise when the four circumstances listed below are met simultaneously:

a) **Mutual Exclusion:**
   A process can only use a resource, such as a printer or a file, at a time. Others must wait for a resource to be released if it is being used by one process. Delays and impasses may result from this.
b) **Hold & Wait:**
   A process may ask for extra resources while retaining the ones it already has. This implies that if other processes require resources that are already used, they may have to wait for a lengthy time, which could result in a stalemate.

c) **No Preemption:**

A resource cannot be taken away by force once it has been seized via a process. The resource must be released by itself. Both processes may become stuck indefinitely if one is waiting for another resource to continue, but that resource is being held by another waiting process.

d) **Circular Wait:**

It occurs when a sequence of processes waits in a circle, each of which requires a resource that is held by the process behind it. All processes stay stalled because nobody can advance.

Deadlock detection:

- **Resource Allocation Graph (RAG)**

  This approach makes use of a graph representation in which resources and processes are nodes and allocation and request relationships are represented by edges.

  A deadlock is shown by a cycle in the graph.Ideal for systems that have a single resource instance.

- **The Wait-for Graph (WFG)**

  a streamlined RAG in which processes waiting on other processes are represented by edges and resources are eliminated.

  Deadlock is confirmed by a cycle in the WFG.WFG is a deadlock detection that is more effective than RAG.

Deadlock Handling & Prevention:

A. Deadlock Ignorance
B. Deadlock Prevention

Mutual Exclusion

Some resources (like printers) can't be shared, so mutual exclusion can't be avoided for them.

Hold and Wait

Don't let processes hold one resource and wait for others**.**
Ask for all needed resources at once**, or** release all before requesting more

No Preemption

If a process can't get all it needs, take away its resources and let others use them. It must try again later when everything it needs is free.

Circular Wait

Assign a number to each resource and force processes to request in order (e.g., 1 → 2 → 3).This breaks the circular chain needed for deadlock.

C. Deadlock Avoidance Algorithms

1. Resource Allocation Strategy

A **Resource Allocation Strategy** is a set of rules or methods the **operating system uses to manage resources** (like CPU, memory, printers, etc.) **among processes** to ensure smooth operation and avoid issues like **deadlocks**.

a) **Main Strategies::**
The system avoids one or more of the four necessary conditions for deadlock. Like NO hold & wait, preemption, circular wait preemption

b) **Deadlock Avoidance (e.g., Banker's Algorithm):**
The system checks every resource request and only approves it if it keeps the system in a safe state**.**

c) Deadlock Detection and Recovery:

The system detects deadlocks (e.g., using a RAG) and fixes them by:

- Killing processes

- Forcing processes to release resources

a) Banker's Algorithm Modification:

It helps to determine if a process can safely be given resources without causing a deadlock. It checks whether giving a process resources would put the system in a safe state or unsafe state**.** This is how it operates:

a) First Step:

- Available resources: Resources not allocated to any process.
- Max resources each process may need: The maximum amount of resources a process may request.
- Current allocation: The resources currently allocated to each process.
- Need matrix: This shows the remaining resources that each process needs to complete.

b) Requesting Resources:

A process asks for some resources. The system checks if granting this request will lead to a safe state. It checks if all processes can eventually finish (without running out of resources).

c) Decision:

If giving the resources still allows the system to be in a safe state, then the request is granted.

Results \ Conclusion:

To sum up, deadlock is a serious problem with operating systems that arises when processes become stopped while waiting for resources that are held by other processes. System stability and performance may be significantly impacted. To effectively manage deadlocks, one must comprehend the four basic conditions: mutual exclusion, hold and wait, no preemption, and circular wait. By using preventive, avoidance, detection, and recovery techniques, systems can manage deadlocks. Deadlock is reduced by the use of strategies for allocating resources and algorithms like the Banker's Algorithm. Although it might not always be feasible to completely eradicate it, a well-designed system can greatly lessen its effects, guaranteeing seamless and effective process execution.

References:

1. https://www.academia.edu/35457722/Various_Methods_for_Solving_Deadlock_Problems_in_Operating_Systems_A_Review
2. https://www.jetir.org/papers/JETIR2404776.pdf
3. https://ijirt.org/publishedpaper/IJIRT100434_PAPER.pdf
4. https://codex.cs.yale.edu/avi/os-book/OS8/os8c/slide-dir/PDF-dir/ch7.pdf