Deadlock Handling

PBL Activity – Computer Systems

Done By :

O.Roshini-23EG107E41

P.Asritha-23EG107E42

P.Sai Ram Nihal-23EG107E45

R.Akshitha-23EG107E51

Branch/Section:

Aiml-E

College/University:

Anurag University

Date:

22/9/25

Abstract

Deadlock handling plays a vital role in maintaining system stability and performance in multiprogramming and multitasking environments. This project aims to simulate and analyze deadlock situations that occur when multiple processes compete for shared resources, resulting in a state where none of them can proceed. The simulation demonstrates all four necessary conditions of deadlock — mutual exclusion, hold and wait, no preemtion and circular wait — providing a clear understanding of how such situations arise in operating systems.

To handle these conditions, the project implements effective techniques such as the Banker's Algorithm for deadlock avoidance, which ensures safe resource allocation by predicting possible unsafe states, and Deadlock Detection and Recovery mechanisms that identify and resolve deadlocks when they occur. The system allows users to visualize process and resource interactions dynamically, showing step-by-step how deadlocks form and how different strategies can prevent or resolve them.

Additionally, real-world case studies of resource allocation in computing environments, such as CPU scheduling, database locking, and memory management, are examined to relate the simulation to practical scenarios. The main objective of this project is to help students and researchers understand the theoretical and practical aspects of deadlocks, enabling them to design more efficient and reliable systems that can prevent or recover from resource contention issues.

Introduction

In modern operating systems, multiple processes often run simultaneously and compete for limited system resources such as memory, CPU time, and I/O devices. When processes hold certain resources while waiting for others that are already occupied, a situation called deadlock may occur. In a deadlock, none of the involved processes can proceed, leading to a system halt and reduced efficiency.

Understanding and handling deadlocks is a crucial aspect of process and resource management. Deadlocks can arise in various environments like databases, file systems, and distributed systems where concurrent access to resources is common. To ensure smooth operation, operating systems employ different strategies such as deadlock prevention, avoidance, detection, and recovery**.**

This project focuses on simulating deadlock conditions to visualize how they occur and exploring techniques to handle them effectively. It implements algorithms like the Banker's Algorithm for deadlock avoidance and Deadlock Detection & Recovery methods to demonstrate practical solutions. Through visual simulation and case studies, the project provides an in-depth understanding of resource allocation, system safety states, and recovery mechanisms—making it a valuable tool for learning and research in operating system concepts.

Key Study Areas

Deadlock Concepts: Understanding causes and conditions of deadlock.

Resource Allocation: Studying how improper allocation leads to deadlocks.

Prevention Techniques: Methods to avoid deadlock occurrence.

Banker's Algorithm: Implementing deadlock avoidance strategy.

Detection & Recovery: Identifying and resolving deadlocks.

Simulation: Demonstrating deadlock scenarios and solutions.

Case Studies: Analyzing real-world resource allocation problems.

Methodology

1. **Problem Identification:**
   Study how deadlocks occur in operating systems due to improper resource allocation among multiple processes.
2. **System Design:**
   Design a simulation model that includes processes, resources, and allocation requests to represent real-world deadlock scenarios.
3. **Deadlock Simulation:**
   Create situations where the four deadlock conditions (mutual exclusion, hold and wait, no preemption, circular wait) are satisfied to demonstrate a deadlock.
4. **Implementation of Algorithms:**
   - Apply **Banker's Algorithm** to avoid unsafe states and ensure safe resource allocation.
   - Implement **Deadlock Detection and Recovery** methods to identify and resolve deadlocks dynamically.
5. **Testing and Analysis:**
   Run different test cases to observe how each algorithm performs under various conditions and compare their efficiency.
6. **Case Study Evaluation:**
   Include real-world examples to analyze how deadlocks affect systems like databases or scheduling processes.
7. **Results and Conclusion:**
   Evaluate the simulation results to understand the effectiveness of different deadlock handling techniques and conclude with best practices.

Result

- The program stops responding after these print statements because both threads are stuck waiting.
- Thread 1 holds `lock1` and waits for `lock2`.
- Thread 2 holds `lock2` and waits for `lock1`.

```
t2.join()

Thread 1: trying to acquire lock1
Thread 1: acquired lock1
Thread 2: trying to acquire lock2
Thread 2: acquired lock2
Thread 1: trying to acquire lock2
Thread 2: trying to acquire lock1
```

Explanation:

1. **Thread 1** acquires **lock1** and then tries to get **lock2**.
2. **Thread 2** acquires **lock2** and then tries to get **lock1**.