Misr University for Science and Technology

College of Business, Economics, and Information system

Business Administration School

(English Department)

Research Submitted by

Student Name: Ahmed Yasser Mohamed

Student ID: 200028671

Introduction

Deadlocks occur when a set of processes are blocked because each holds a resource and waits for another resource held by another process in the set. This results in a standstill where no progress can be made.

Key Conditions for Deadlocks

1. **Mutual Exclusion**: Only one process can use a resource at a time.
2. **Hold and Wait**: A process holds at least one resource while waiting for others.
3. **No Preemption**: Resources cannot be forcibly taken from a process.
4. **Circular Wait**: A cycle exists where each process waits for a resource held by the next.

Handling Deadlocks

There are three main approaches:

1. **Prevention**: Design the system to ensure at least one of the four conditions cannot hold (e.g., enforcing a strict resource request order).
2. **Avoidance**: Use algorithms like the Banker's Algorithm to dynamically check if allocating resources would lead to an unsafe state.
3. **Detection and Recovery**: Allow deadlocks to occur, detect them using resource allocation graphs or detection algorithms, and recover by terminating processes or preempting resources.

Practical Examples

- **Resource Allocation Graphs**: Visual tools to detect cycles indicating deadlocks.
- **Banker's Algorithm**: A mathematical model to ensure safe resource allocation.
- **Java Deadlocks**: Common in multithreaded programs when threads acquire locks in inconsistent orders.

Conclusion

Deadlocks are a critical issue in concurrent systems. While prevention and avoidance are proactive, detection and recovery offer reactive solutions.

Most operating systems, like UNIX, ignore deadlocks due to their rarity, but understanding these concepts is essential for robust system design.