Deadlocks

A **deadlock** is a situation in computer systems where a group of processes are stuck in a state where none can proceed because each process is waiting for a resource that another process in the group is holding. Deadlocks typically occur in multi-tasking or multi-threaded environments when processes or threads compete for shared resources.

Key Conditions for Deadlock

A deadlock can arise if all the following conditions hold simultaneously (commonly called the Coffman conditions):

1. **Mutual Exclusion**: At least one resource must be held in a non-shareable mode, meaning only one process can use it at a time.
2. **Hold and Wait**: A process holding at least one resource is waiting to acquire additional resources that are currently held by other processes.
3. **No Preemption**: Resources cannot be forcibly taken away from a process. They must be released voluntarily by the holding process.
4. **Circular Wait**: There exists a set of processes {P1, P2, ..., Pn} such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3, and so on, with Pn waiting for a resource held by P1.

Example of a Deadlock

Imagine two processes, P1P1 and P2P2, and two resources, R1R1 and R2R2:

1. P1P1 holds R1R1 and is waiting to acquire R2R2.
2. P2P2 holds R2R2 and is waiting to acquire R1R1.

Neither process can proceed because they are waiting for each other to release a resource, resulting in a deadlock.

Deadlock Prevention

To prevent deadlocks, at least one of the Coffman conditions must be violated. Techniques include:

1. **Deny Mutual Exclusion**: Make resources shareable (not always practical, e.g., for printers).
2. **Deny Hold and Wait**: Require processes to request all resources at once.
3. **Allow Preemption**: Allow resources to be forcibly taken from a process.
4. **Deny Circular Wait**: Impose an ordering on resource acquisition to prevent cycles.

Deadlock Detection and Recovery

If prevention isn't feasible, systems can:

1. **Detect Deadlocks**: Use resource allocation graphs or algorithms to detect cycles.
2. **Recover from Deadlocks**: Abort processes or preempt resources to break the cycle.

Practical Applications

Deadlocks are a key concern in:

- **Operating Systems**: Resource sharing like CPU, memory, files.
- **Databases**: Transactions waiting for locks.
- **Distributed Systems**: Network communications or distributed transactions.

Understanding deadlocks is crucial for designing systems that efficiently manage concurrency without becoming unresponsive.