Concepts of Operating Systems

- Vineela

Session 9:

Deadlock

- Necessary conditions of deadlock
- Deadlock prevention and avoidance
- Semaphore
- Mutex
- Producer consumer problem
- Dead-lock vs Starvation

Deadlock

- Deadlock is a situation in which two or more processes require resources to complete their execution, but those resources are held by another process., due to which the execution of the process is not completed.
- **For example** - suppose there are two friends and both want to play computer games, due to which they fight. One has a remote control, and the other has a CD of games.
- Due to this, neither of the two friends can play, but neither of them is ready to cooperate. This situation is called deadlock.

Necessary conditions of deadlock

- **Mutual Exclusion -** At least one resource must be held in a **non-shareable mode**. Only one process can use the resource at any given time.

  Example: A printer cannot be used by two processes at once.
- **Hold and Wait -** A process is holding at least one resource and is **waiting to acquire additional resources** that are currently being held by other processes.

  Example: Process A holds a file and waits for a printer held by Process B.
- **No Preemption**: Resources **cannot be forcibly taken** from the processes holding them; they must be released voluntarily.

  Example: You can't just take a file from a process — it must release it on its own.
- **Circular Wait**: A set of processes $\{P_1, P_2, ..., P_□\}$ exists such that **P_1 is waiting for a resource held by P_2, P_2 is waiting for a resource held by P_3, ..., and $P_□$ is waiting for a resource held by P_1**, forming a circular chain.
  - **Example**: P1 → P2 → P3 → P1 (circular dependency).

Deadlock prevention and avoidance

NOTE : Deadlock can occur **only if all four** of these conditions hold **simultaneously**. If **even one** of them is prevented or broken, **deadlock cannot occur**.

- Deadlock prevention and avoidance are the strategies used in operating systems to ensure that processes do not get stuck waiting for resources indefinitely.

Deadlock Prevention

Deadlock prevention works by eliminating one of the four necessary conditions for deadlock:

1. **Mutual Exclusion** – Ensuring that resources are shared whenever possible.
2. **Hold and Wait** – Requiring processes to request all resources at once, preventing them from holding some while waiting for others.
3. **No Preemption** – Allowing the system to forcibly take resources from a process if needed.
4. **Circular Wait** – Imposing an ordering on resource requests to prevent circular dependencies.

Deadlock Avoidance

- Deadlock avoidance, on the other hand, does not eliminate conditions but ensures that the system never enters an unsafe state. It relies on algorithms like:
  - **Banker's Algorithm** – Ensures that resource allocation always leaves the system in a safe state.
  - **Resource Allocation Graph** – Tracks dependencies and prevents cycles from forming.

Semaphore

- A Semaphore is a synchronization mechanism used in operating systems to manage access **to shared resources and prevent race conditions** in concurrent processes.

- Semaphores help ensure **mutual exclusion**, **process synchronization**, and **deadlock prevention** in multi-threaded environments

Types of Semaphores

1. Counting Semaphore – Allows multiple processes to access a resource up to a certain limit.
2. Binary Semaphore – Works like a lock, allowing only one process at a time.

Operations

- Wait (P operation) – Decreases the semaphore value; if it's already zero, the process waits.
- Signal (V operation) – Increases the semaphore value, allowing waiting processes to proceed.

Mutex

- A mutex (short for **mutual exclusion**) is a synchronization mechanism used in operating systems **to control access to shared resources in a multi-threaded environment.**

How Mutex Works

- A **mutex lock** ensures that only one thread can access a critical section at a time.
- When a thread acquires the mutex, other threads must wait until it is released.
- Once the thread finishes its operation, it releases the mutex, allowing another thread to proceed.

Types of Mutex

- Recursive Mutex – Allows the same thread to lock the mutex multiple times without causing a deadlock.
- Error-Checking Mutex – Prevents a thread from acquiring a mutex it already holds, helping detect programming errors.

Mutex vs Semaphore

- Mutex is a binary lock (either locked or unlocked), while semaphores can allow multiple threads to access a resource up to a defined limit.
- Mutex is used for mutual exclusion, whereas semaphores can be used for synchronization.

Producer-Consumer Problem

- The Producer-Consumer Problem is a classic synchronization problem in operating systems that illustrates the need for proper coordination between processes or threads that share a common resource, such as a buffer.

Problem Overview : There are two types of processes:

- Producer: Generates data and puts it into a shared buffer.
- Consumer: Takes data from the shared buffer and processes it.

Problem:

- The buffer has limited size.
- The producer must wait if the buffer is full.
- The consumer must wait if the buffer is empty.
- Both must not access the buffer at the same time (to avoid race conditions).

Mutual Exclusion	:	Ensure only one process accesses the buffer at a time.
Synchronization	:	Coordinate producer and consumer to prevent overfilling or underflow.
Critical Section	:	The part of the code where shared resources are accessed.

Solution Approaches : Using Semaphores

- Print Spooler: Produces print jobs and adds them to a queue; printer (consumer) processes them.
- Data Streaming:Video/audio producer generates frames/samples; consumer renders them.

Dead-lock vs Starvation

Feature	Deadlock	Starvation
Definition	Circular waiting with no progress	Process waits indefinitely
System state	Entire group of processes are stuck	Other processes may continue
Cause	Circular wait on resources	Biased resource allocation (e.g. priority)
Detection	Detectable using algorithms	Harder to detect
Resolution	Needs deadlock recovery or prevention	Needs fair scheduling (e.g., aging)
Processes affected	All in the cycle	One or few low-priority processes

- Deadlock = No one can proceed.
- Starvation = One can't proceed, but others can.
- Deadlock is often more critical and harder to resolve than starvation.
- Starvation can happen without deadlock, but deadlock often causes starvation.