**Basics of Operating Systems**

**Intro:**

**Definition:**

An OS is system software that manages computer hardware and software resources and provides common services for computer programs.

**Functions:**

**Process Management:** Manages the execution of processes, including creation, scheduling, and termination.

**Memory Management:** Handles allocation and deallocation of memory spaces.

**File System Management:** Manages files and directories, including storage, retrieval, and organization.

**Device Management:** Controls peripheral devices through their drivers.

**Security and Access Control:** Ensures authorized access to data and resources.

**Types of OS:**

**Batch OS:** Execute batches of jobs without user interaction. Suitable for repetitive tasks.

**Time-Sharing OS:** Allow multiple users to use the system simultaneously by quickly switching between users. Provides interactive computing.

**Distributed OS:** Manage a group of independent computers and make them appear to be a single coherent system.

**Real-Time OS:** Provide immediate processing and responses to inputs.

- **Hard Real-Time Systems:** Strict timing constraints.

- **Soft Real-Time Systems:** Less stringent timing constraints.

**Network OS:** Provide services to computers connected in a network, enabling resource sharing.

**Mobile OS:** Designed for mobile devices (e.g., iOS, Android).

**Process Creation and Termination**

**Process Creation:**

**Fork System Call:**

Creates a new process by duplicating an existing process.

The new process (child) is a copy of the parent process.

**exec System Call:**

Replaces the process’s memory space with a new program.

**Steps:**

Assign a unique process identifier (PID).

Allocate memory for the process.

Initialize process control block (PCB).

Set up appropriate linkages (e.g., parent-child relationships).

Place the process in the ready queue.

**Process Termination:**

**Normal Termination:** Process completes its execution successfully.

**Abnormal Termination:** Process is terminated due to errors or external signals (e.g., divide by zero, kill signal).

**Steps:**

Release allocated resources.

Update process state to terminated.

Remove process from scheduling queues.

Inform the parent process (if any) of the termination.

**Pre-emptive Scheduling Techniques**

**Definition:**

Scheduling method where the operating system can pre-empt a running process to assign the CPU to another process.

**Types:**

**Round Robin (RR):**

Each process is assigned a fixed time slice (quantum).

Processes are scheduled in a circular order, providing fair time-sharing.

**Shortest Remaining Time First (SRTF):**

Pre-empts the running process if a new process arrives with a shorter remaining execution time.

Aims to minimize waiting time.

**Priority Scheduling:**

Processes are scheduled based on priority.

The OS pre-empts the running process if a higher-priority process arrives.

Can lead to starvation of low-priority processes.

**Multilevel Queue Scheduling:**

Multiple queues based on priority levels, each with its own scheduling algorithm.

Pre-empts processes between different queues based on priority.

**Multilevel Feedback Queue Scheduling:**

Processes can move between queues based on their behaviour and requirements.

Pre-empts processes to provide flexibility and responsiveness.

**Process Scheduling**

**Non-pre-emptive Scheduling Techniques**

**Definition:**

Once the CPU is assigned to a process, it cannot be taken away until the process completes its execution or voluntarily releases the CPU.

**Types:**

**First-Come, First-Served (FCFS):**

Processes are scheduled in the order they arrive.

Simple but can lead to long average waiting times, particularly for processes arriving late.

**Shortest Job Next (SJN):**

Selects the process with the shortest execution time.

Optimal for minimizing average waiting time but requires knowledge of the process execution times.

**Priority Scheduling:**

Processes are scheduled based on priority.

High-priority processes are selected first.

Can lead to starvation of low-priority processes unless aging is implemented.

**Race Conditions**

**Definition:**

A situation where the behaviour and outcome of software depend on the sequence or timing of uncontrollable events, such as the order in which processes access shared resources.

**Causes:**

Occur when multiple processes access and manipulate shared data concurrently, leading to inconsistent or unpredictable results.

**Prevention Techniques:**

**Mutexes:**

Mutual exclusion locks to ensure that only one process can access a resource at a time.

**Semaphores:**

Synchronization primitives that signal and wait to control access to resources.

**Monitors:**

High-level synchronization constructs that encapsulate shared variables and operations, allowing only one process to execute an operation at a time.

**Atomic Operations:**

Operations that complete in a single step without interruption, ensuring consistency of shared data.

**Threads and Multithreading**

**Basics of Threads:**

**Thread:** The smallest unit of a process that can be scheduled and executed independently.

**Process:** Contains at least one thread.

**Multithreading:** Allows multiple threads to exist within a single process, sharing resources but able to run independently.

**Thread Life Cycle:** New -> Runnable -> Running -> Blocked -> Dead.

**Advantages of Multithreading:**

**Resource Sharing:**

Threads share the same memory space, which makes communication faster.

**Responsiveness:**

Applications remain responsive to user input by running separate threads for different tasks.

**Utilization of Multiprocessor Architectures:**

Threads can run on multiple processors, enhancing performance.

**Disadvantages of Multithreading**

**Complexity:** Writing and managing multithreaded applications can be complex.

**Concurrency Issues:** Problems like deadlocks, race conditions, and thread starvation can occur.

**Implementing Threads**

**User-Level Threads:** Managed by user-level libraries, fast context switching, but lack kernel support.

**Kernel-Level Threads:** Managed by the OS kernel, slower context switching, but better performance in multi-core processors.

**Memory Management**

**Contiguous Memory Allocation:**

**Fixed Partitioning:**

Memory is divided into fixed-size partitions.

Simple but can lead to internal fragmentation.

**Dynamic Partitioning:**

Memory is allocated dynamically, leading to external fragmentation and the need for compaction.

**Paging**

**Paging:**

Memory is divided into fixed-size pages, and physical memory is divided into frames of the same size.

**Page Table:**

Maintains the mapping between virtual addresses and physical addresses.

**Advantages:**

Eliminates external fragmentation, allows non-contiguous memory allocation.

**Disadvantages:**

Can introduce overhead due to the need for page table management.

**Segmentation**

**Segmentation:**

Memory is divided into segments based on logical divisions such as functions, objects, or data.

**Segment Table:** Maintains the base and limit addresses of each segment.

**Advantages:**

Provides a logical way of dividing memory, can lead to better utilization.

**Disadvantages:**

Can introduce external fragmentation, and requires complex segment management.

**Virtual Memory**

**Thrashing and Page Faults:**

**Page Fault:**

Occurs when a requested page is not in memory, causing the OS to fetch it from the disk.

**Thrashing:**

A state where excessive paging operations occur, degrading performance.

**Page Replacement Algorithms**

**First-In-First-Out (FIFO):**

Replaces the oldest page. Simple but can lead to Belady's anomaly.

**Least Recently Used (LRU):**

Replaces the page that has not been used for the longest time. Effective but requires complex tracking.

**Optimal Page Replacement:**

Replaces the page that will not be used for the longest time in the future. Ideal but impractical as it requires future knowledge.

**Clock Algorithm:**

A circular list of pages with a reference bit to approximate LRU.

**File Systems**

**Intro:**

**File System (FS):**

A method and data structure that the operating system uses to manage files on a disk or partition.

**Functions of a FS:**

**File Creation and Deletion:**

Allowing users and applications to create and delete files.

**Directory Creation and Deletion:**

Organizing files into directories for easier management.

**Support for Storage Devices:**

Managing different types of storage devices (e.g., HDD, SSD).

**File Manipulation:** Providing methods to read, write, and update files.

**File Protection and Security:** Ensuring only authorized users can access and modify files.

**File System Implementation (FSI)**

- **File System Structure:**

**Boot Block:** Contains the bootstrap loader program.

**Superblock:** Contains information about the file system (e.g., size, number of files).

**Inode Table**: Contains metadata about files.

**Data Blocks:** Store the actual file data.

**File Allocation Methods:**

**Contiguous Allocation:**

Files are stored in contiguous blocks. Simple but can lead to fragmentation.

**Linked Allocation:**

Each file is a linked list of blocks. No fragmentation, but can be slow.

**Indexed Allocation:**

Uses an index block to keep track of all blocks belonging to a file. Balances speed and space.

**Directory Implementation:**

**Single-Level Directory:**

All files are contained in a single directory. Simple but impractical for large systems.

**Two-Level Directory:**

Separate directory for each user. Avoids name collisions but still limited.

**Tree-Structured Directory:**

Hierarchical directory structure. Most common and flexible.

**Acyclic-Graph Directory:**

Allows shared subdirectories and files. More complex management.

**Free Space Management:**

**Bit Vector:** Uses a bitmap to track free blocks.

**Linked List:** Links together all free blocks.

**Grouping:** Tracks free blocks in groups.

**Counting:** Tracks the number of contiguous free blocks.

**Concurrency and Synchronization**

**Deadlocks**

**Definition:**

A situation in a multitasking environment where two or more processes are unable to proceed because each is waiting for the other to release a resource.

**Deadlock Prevention:**

- Ensure at least one of the following conditions is impossible:

**Mutual Exclusion:** Not all resources are sharable.

**Hold and Wait:** Processes are not allowed to hold one resource while waiting for another.

**No Pre-emption:** Resources cannot be forcibly taken from processes.

**Circular Wait:** Impose an ordering on resource types and ensure processes request resources in increasing order.

**Deadlock Avoidance**

**Banker's Algorithm:**

Ensures that a system will remain in a safe state by pre-evaluating resource allocation requests.

**Resource Allocation Graph:**

Uses graph theory to detect the possibility of a deadlock.

**Deadlock Detection and Recovery**

**Detection:**

**Wait-For Graph:**

Periodically check for cycles in a graph of resource allocations.

**Resource Allocation Graph:**

Tracks resource allocation to detect cycles.

**Recovery:**

**Process Termination:** Abort one or more processes to break the deadlock.

**Resource Pre-emption:** Temporarily take resources away from processes and reassign them.

**Semaphores and Mutexes**

**Semaphore:**

A synchronization primitive that can be used to control access to a common resource by multiple processes.

**Binary Semaphore:** Also called a mutex. Only two states (locked/unlocked).

**Counting Semaphore:** Allows more than one resource instance.

**Operations:**

`wait()` (decrements the semaphore) `signal()` (increments the semaphore).

**Mutex (Mutual Exclusion):**

A locking mechanism to ensure that only one thread can access a resource at a time.

**Lock/Unlock:** Operations to gain and release control over the resource.

**Monitors and Condition Variables**

**Monitor:**

A high-level synchronization construct that allows safe access to shared data.

**Condition Variables:**

Used within monitors to manage complex synchronization.

**Operations:** `

wait( ): (releases the monitor and waits for a condition)

signal( ): (wakes up a waiting thread).

**Advanced Memory Management**

**Cache Memory**

**Definition**:

A small, fast memory located close to the CPU to store frequently accessed data and instructions.

**Purpose:** To reduce the average time to access data from the main memory.

**Types of Cache:**

**L1 Cache:** Located on the CPU chip, smallest and fastest.

**L2 Cache:** Larger than L1, may be on the CPU or a separate chip.

**L3 Cache:** Shared among cores in multi-core processors, larger but slower than L1 and L2.

**Cache Replacement Policies**

**First-In-First-Out (FIFO):**

Replaces the oldest cache line. Simple but not always efficient.

**Least Recently Used (LRU):**

Replaces the cache line that has not been used for the longest time.

More efficient but requires tracking of usage.

**Random Replacement:**

Replaces a randomly selected cache line. Simple but can be inefficient.

**Least Frequently Used (LFU):**

Replaces the cache line that is used least frequently.

Requires counting usage frequency.

**Adaptive Replacement Cache (ARC):**

Balances between LRU and LFU, adjusting dynamically to the workload.

**OS Security**

**Access Control and Authentication**

**Access Control:**

Mechanisms to ensure only authorized users can access resources.

**Discretionary Access Control (DAC):**

Access rights are assigned by the owner.

**Mandatory Access Control (MAC):**

Access rights are determined by the system based on policies.

**Role-Based Access Control (RBAC):**

Access rights are based on the roles assigned to users.

**Authentication:** Verifying the identity of a user or process.

**Password-Based Authentication:** Using passwords to authenticate users.

**Multi-Factor Authentication (MFA):**

Combines two or more independent credentials (e.g., password and OTP).

**Biometric Authentication:**

Uses biological traits (e.g., fingerprints, facial recognition).

**Secure OS Design and Implementation**

**Principles:**

**Least Privilege:** Users and processes operate with the minimum privileges necessary.

**Défense in Depth:** Multiple layers of security controls.

**Fail-Safe Defaults:** Access decisions should default to denial.

**Security Features:**

**User Authentication:** Ensuring only authorized users can access the system.

**Access Control Mechanisms:** DAC, MAC, RBAC.

**Auditing and Logging:** Tracking and recording system activities for monitoring and forensic analysis.

**Encryption:** Protecting data at rest and in transit.

**Malware and Défense Mechanisms**

**Malware:** Malicious software designed to disrupt, damage, or gain unauthorized access to systems.

**Types of Malwares:**

**Viruses:** Attach to legitimate programs and spread when executed.

**Worms:** Self-replicating malware that spreads across networks.

**Trojan Horses:** Disguised as legitimate software but perform malicious activities.

**Ransomware:** Encrypts data and demands payment for decryption.

**Spyware:** Collects information without user consent.

**Adware:** Displays unwanted advertisements.

**Défense Mechanisms:**

**Antivirus Software:** Detects and removes malware.

**Firewalls**: Monitors and controls incoming and outgoing network traffic.

**Intrusion Detection Systems (IDS):** Monitors network or system activities for malicious activities.

**Patch Management:** Regularly updating software to fix vulnerabilities.

**User Education and Awareness:** Training users on safe computing practices.

**Intro to DBMS and RDBMS**

**DBMS (Database Management System)**

**Definition:** A software system that uses a standard method of cataloguing, retrieving, and running queries on data.

**Functions:**

Data storage, retrieval, and update. User and system administration.

Data security and integrity. Backup and recovery.

**Types:**

**Hierarchical DBMS:** Data is organized in a tree-like structure.

**Network DBMS:** Uses a graph structure to organize data.

**Relational DBMS (RDBMS):** Data is stored in tables (relations).

**RDBMS (Relational Database Management System)**

**Definition:** A type of DBMS that stores data in a tabular form and allows relationships between tables through keys.

**Characteristics:**

Data is organized in tables.

Supports SQL for querying and managing data.

Ensures ACID properties (Atomicity, Consistency, Isolation, Durability).

**Examples**: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

**SQL vs NoSQL**

**SQL (Structured Query Language)**

**Definition:** A standard programming language for managing and manipulating relational databases.

**Characteristics:**

Schema-based, structured data. Relational data model.

Supports complex queries and transactions.

**Examples:** MySQL, PostgreSQL, SQLite.

**NoSQL (Not Only SQL)**

**Definition:** A class of database management systems that do not adhere strictly to a relational model.

**Characteristics:**

Schema-less, flexible data model.

Handles unstructured or semi-structured data.

High scalability and performance for large datasets.

**Types:**

**Document Store:** MongoDB, CouchDB.

**Key-Value Store**: Redis, DynamoDB.

**Column Store:** Cassandra, HBase.

**Graph Store:** Neo4j, ArangoDB.

**SQL Queries: DDL, DML, and DCL**

**DDL (Data Definition Language)**

**Purpose:** Define and modify database structure.

**Commands:**

**CREATE:** Creates a new table, view, or database.

**ALTER:** Modifies an existing database object.

**DROP:** Deletes a table, view, or database.

**TRUNCATE:** Removes all rows from a table without logging individual row deletions.

**DML (Data Manipulation Language)**

**Purpose:** Manipulate data within existing structures.

**Commands:**

**SELECT:** Retrieves data from a database.

**INSERT:** Adds new rows to a table.

**UPDATE:** Modifies existing data within a table.

**DELETE:** Removes rows from a table.

**DCL (Data Control Language)**

**Purpose:** Control access to data.

**Commands:**

**GRANT:** Gives user access privileges to the database.

**REVOKE:** Removes user access privileges.

**SQL Queries: TCL and DQL**

**TCL (Transaction Control Language)**

**Purpose:** Manage transactions in a database.

**Commands:**

**COMMIT:** Saves all changes made in the current transaction.

**ROLLBACK:** Undoes changes made in the current transaction.

**SAVEPOINT:** Sets a save point within a transaction for partial rollbacks.

**DQL (Data Query Language)**

**Purpose:** Query data from a database.

**Commands:**

**SELECT:** Retrieves data from one or more tables.

**Functional Dependency, Normalization, and Normal Forms**

**Functional Dependency**

**Definition:**

A relationship between two attributes, typically between a key and non-key attribute, such that the value of one attribute determines the value of the other.

**Notation:**

\( A \rightarrow B \) (Attribute B is functionally dependent on Attribute A).

**Normalization**

**Purpose:** Organize data to reduce redundancy and improve data integrity.

**Process:** Decomposing a table into smaller tables and defining relationships between them.

**Normal Forms**

**1NF (First Normal Form):**

Ensures each column contains atomic (indivisible) values.

Each column contains values of a single type.

**2NF (Second Normal Form):**

Meets all requirements of 1NF.

All non-key attributes are fully functional dependent on the primary key.

**3NF (Third Normal Form):**

Meets all requirements of 2NF.

There are no transitive dependencies (non-key attribute dependent on another non-key attribute).

**BCNF (Boyce-Codd Normal Form):**

Meets all requirements of 3NF.

For every functional dependency \( A \rightarrow B \), A should be a super key.