Day 2 Tasks

**Task 1: Understanding Process Creation and Scheduling**

**Objective**: Learn how processes are created and scheduled in an operating system. **Steps**:

* Write a simple hello world program in any language , say Python.
* Compile and run the program, observing the process ID (PID) and other information using system tools like ps or top in Linux.
* Use ps or other process monitoring tools to check the state of the process(ready ,running) during execution.
* Create multiple processes (either by running the program multiple times or use multitasking in Python.). Observe how the OS handles these processes and their scheduling.

**Question**: How does the operating system decide which process gets CPU time? What role does the process scheduler play in this?

**Ans** :The os use a Process Scheduler to allocate CPU time to various processes. When a process is created, it is assigned a process id(pid), and its state is initially set to new.

The scheduler moves the process through different states like, ready, running, blocked and terminated.

The scheduling decision is based on FCFS or round robin. So, OS use these algorithm to decide which process to execute next, balancing fairness and efficiency in resource allocation.

**Task 2: Exploring Thread Management and Concurrency**

**Objective**: Understand the concept of multithreading and how concurrency works in an operating system.

**Steps**:

* Write a simple program that uses multiple threads to perform tasks concurrently (e.g., one thread for downloading a file, another for processing data).
* Use thread management tools to monitor and manage these threads (e.g., top, htop, or task manager in your OS).
* Use synchronization mechanisms like semaphores or mutexes to manage shared resources between threads.
* Test how threads interact with each other and ensure data consistency is maintained with synchronization.

**Question**: How does multithreading improve system performance, and what challenges does it introduce in terms of data consistency and synchronization?

**Ans :** Multithreading improves performance by allowing the CPU to execute multiple threads concurrently. This makes better use of multicore processors and increases throughput.

Concurrency is the management of multiple tasks, but not necessarily simultaneously.

Synchronization mechanisms like mutexes and semaphores are used to avoid conflicts when thread access shared resources. Without proper synchronization , data inconsistency and race condition may occur, which can led to incorrect result and crashes.

**Task 3: Investigating Deadlock in Multithreading**

**Objective**: Learn how deadlock occurs in multithreading environments and how to detect and resolve it.

**Steps**:

* Write a simple program with two threads where each thread locks a resource, then tries to acquire the lock on the other's resource, causing a deadlock.
* Use a debugger or logging to trace the execution and confirm that the program is deadlocked.
* Modify the program to resolve the deadlock by introducing timeouts or changing the locking order

**Question :** What conditions are necessary for a deadlock to occur? How can deadlock be avoided or resolved in multithreaded programs?

**Ans :** So, conditions are :

* **Mutual Exclusion:** Resources can only be held by one thread at a time.
* **Hold and Wait :** A thread holds at least one resource and is waiting to acquire other resources held by other threads.
* **No Pre-emption:** Resources cannot be forcibly taken from a thread.
* **Circular Wait**: A set of threads is waiting for resources in a circular manner.

To resolve deadlock, you can use methods like:

* **Deadlock prevention** (by eliminating one of the condition)
* **Deadlock detection** and **recovery** (detecting when a deadlock occurs and recovering from it by terminating or rolling back processes).

**Task 4: Understanding I/O Operations and Buffering**

**Objective**: Explore how the OS handles I/O operations using buffers and how data is managed between processes and devices.

**Steps**:

* Write a program that reads from and writes to a file, ensuring that the data is buffered for efficiency.
* Use system tools like vmstat or iostat to monitor I/O operations and buffer usage while your program is running.
* Introduce large files to your program and observe how the OS handles large-scale I/O operations, paying attention to any performance impacts.

**Question**: How do I/O buffers improve the efficiency of file operations? What impact do buffer sizes have on the system's performance during I/O operations?

**Ans:** I/O buffering improves performance by temporarily storing data in memory before it is written to or read from an external device. This reduce the number of read/write operations required and optimizes the efficiency of I\O.

Buffer sizes can significantly impact performance; for eg, larger buffers can handle more data at once but may introduce latency.In general, buffering allows the system to continue performing other tasks while waiting for I/O operations to complete, improving overall system throughput.

**Task 5: Analysing I/O Scheduling Algorithms**

**Objective**: Understand how different I/O scheduling algorithms impact system performance. **Steps**:

* Use a Linux-based system and run multiple I/O intensive tasks like copying large files or running disk benchmarks.
* Analyse I/O scheduling using the iotop or dstat command to monitor disk activity and the types of scheduling algorithms in use (e.g., CFQ, Deadline, or NOOP).
* Test different algorithms and measure the time taken for I/O operations, then compare the results.

**Question**: How do different I/O scheduling algorithms affect disk performance? What are the advantages and disadvantages of each algorithm in different workloads?

I/O scheduling algorithm determine the order in which I/O requests are processed. Each algorithm has its strength.

* **FCFS**: Simple but may cause long waiting times for disk access.
* **SSTF** : prioritizes I/O requests closest to the current disk head position, reducing seek time but can cause starvation of distant requests.
* **Elevator(SCAN):**  The disk head moves in one direction until it reaches the end of the disk, then reverses direction. This minimizes seek time but can cause delays in some cases.
* **Deadline Scheduling:** Prioritizes I/O requests with strict deadlines.

**Task 6: Exploring the OSI Model - Layers and Functions**

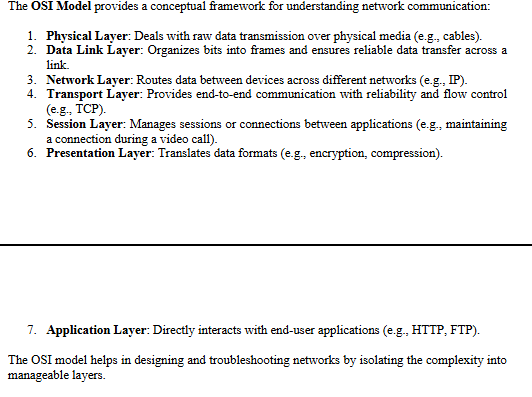
**Objective**: Learn about the OSI model and the role each layer plays in network communication. **Steps**:

1. Research and create a diagram of the OSI model, labeling each of the seven layers and describing their functions.

2. Set up a basic network communication system using a protocol like **ping** (Layer 3) or **HTTP** (Layer 7).

3. Use network tools like traceroute or tcpdump to capture packets and identify which OSI layers are involved during communication.

**Question**: How does the OSI model help in understanding network communication? What role does each layer play in ensuring data is transmitted across the network?



**Task 7: Testing Transport Layer Protocols (TCP/UDP)**

**Objective**: Understand the differences between TCP and UDP and how they function in the transport layer

. **Steps**:

1. Use the netstat command to observe network connections and identify which ones use TCP and which use UDP.

2. Write simple client-server programs using TCP and UDP (e.g., a chat application).

3. Test both programs and observe how TCP guarantees reliability (via handshakes and acknowledgments) while UDP does not.

**Question**: What are the key differences between TCP and UDP in terms of reliability and performance? When would you choose one over the other?

A close up of a text

Description automatically generated

**Task 8: Exploring Linux Process Management**

**Objective**: Understand how processes are managed in Linux, including creation, scheduling, and termination.

**Steps**:

1. Use the ps and top commands to list the running processes on a Linux system.

2. Create and kill processes using the fork() system call or equivalent in a C program.

3. Monitor process states using ps or htop to understand how the Linux kernel schedules processes.

**Question**: How does Linux manage processes in terms of scheduling, and how can you observe and control the state of processes using system tools?

A black text on a white background

Description automatically generated

**Task 9: Linux File System and Permissions**

**Objective**: Learn how Linux organizes files and manages permissions for users. **Steps**:

1. Explore the Linux file system using commands like ls, cd, and df to understand the directory structure and disk usage.

2. Create files and directories, setting various permissions using the chmod, chown, and chgrp commands.

3. Test access control by creating multiple users and groups and modifying file permissions to restrict or allow access.

**Question**: How do file permissions in Linux enhance security? How does the ownership model (user, group, others) control access to resources?

A white background with black text

Description automatically generated

**Task 10: Using Shell Scripting for Process Automation**

**Objective**: Learn how to automate system tasks using Linux shell scripting.

**Steps**:

1. Write a shell script to automate a simple task, such as backing up a directory or renaming files in a batch process.

2. Use system commands like cron to schedule the script to run at a specific time or interval.

3. Test the script and ensure it runs as expected, handling errors or exceptions if necessary.

**Question**: How can shell scripting help automate system management tasks? What advantages does it offer for managing processes and resources on a Linux system?

A white paper with black text

Description automatically generated