Session 1: Introduction to Operating System and Its Terminologies

1. Introduction to Operating System (OS) and Its Terminologies
An Operating System (OS) is software that acts as an interface between hardware and software. It controls computer hardware and provides an easy interface for users to perform their tasks.

Basic Terminologies:

- Operating System (OS): Examples include Windows, Linux, and macOS.
- Process: A running program that utilizes CPU time and memory.
- Thread: A smaller unit of execution within a process.
- System Call: An interface that allows application software to access OS resources.

---

2. Kernel Components and Non-Kernel Components

The Operating System can be divided into two major components:

Kernel Components:

- Kernel: The core part of the OS that manages hardware resources like memory, CPU, and I/O devices. It is considered the "heart" of the OS.
- Memory Management: Determines which process gets how much memory and removes memory from a process when needed.
- Process Management: Handles the creation, scheduling, and termination of processes.
- File System Management: Organizes files on disk and manages file operations.

Non-Kernel Components:

- User Interface (UI): The layer through which users interact with the OS, such as a Graphical User Interface (GUI) or Command-Line Interface (CLI).
- Applications: External programs that perform user tasks, such as web browsers and media players.

---

3. User-space vs Kernel-space

The operating system is divided into two distinct spaces:

Kernel-space:

- Reserved for the operating system's kernel.
- Directly interacts with hardware resources.
- Responsible for high-level operations like CPU scheduling, memory management, and process control.

User-space:

- Area where user applications run.
- Restricted from accessing kernel-space resources directly.
- Applications must use system calls to request hardware resource access through the operating system.

Example: If a program needs access to system memory or disk, it must send a request from user-space to kernel-space via system calls.

---

2. Hardware Interrupts and Interrupt Handlers

Interrupts:

- Signals sent by hardware devices to the CPU when immediate attention is required.
- For example, pressing a keyboard key sends an interrupt to the CPU, notifying it of user action.

Interrupt Handler:

- A special program executed to handle the interrupt.
- Takes actions based on the type of interrupt, such as reading data from an I/O device or processing events.

Example: When a printer completes a job, it sends an interrupt to the CPU. The interrupt handler checks the printer's status and takes appropriate actions.

Summary

- The operating system acts as a bridge between hardware and applications.
- Kernel is the core component that manages hardware resources.
- User-space runs user applications, while Kernel-space handles core OS functionalities.
- Interrupts are hardware signals requiring immediate CPU attention, and interrupt handlers manage these events.

Understanding the OS architecture helps in comprehending how systems work effectively and efficiently.

Session 2

1. Process Management

- Definition: Process management is a crucial function of the operating system that handles the execution of processes, resource allocation, and monitoring.
- What is a Process?
    - A process is a running instance of a program.
    - Each process has a Process Control Block (PCB), which contains key information like:
        - Process ID
        - CPU registers
        - Program counter
        - Memory pointers, etc.

---

2. Process Scheduling

- Determines the sequence of process execution.
- Allocates CPU time to processes based on priorities and system requirements.

Scheduler:

- A part of the operating system responsible for scheduling processes.
- Selects a process for execution and allocates CPU when it is free.

---

3. CPU Scheduling

- Focuses on allocating CPU time to processes in an efficient manner.
- Only one process can use the CPU at a time.
- The goal is to optimize system performance and maximize CPU utilization.
- The CPU Scheduler manages tasks in a queue and decides the execution order.

---

4. Preemptive vs Non-Preemptive Scheduling

Preemptive Scheduling:

- Allows the operating system to interrupt a running process if a higher-priority process needs to execute.
- Example: If a process is using too much CPU time, it can be suspended to allow another process to run.

Non-Preemptive Scheduling:

- Once a process starts execution on the CPU, it cannot be interrupted until it finishes.
- Example: Other processes must wait until the current process completes its task.

---

5. Scheduling Algorithms

First Come First Serve (FCFS):

- Processes are executed in the order they arrive.
- Advantages: Simple to implement.

- Disadvantages: A long process arriving first can cause delays for others (known as the convoy effect).
- Example:
  If processes 1, 2, and 3 arrive in that order, they are executed in the same order.

Round Robin (RR):

- CPU time is divided into fixed time slices.
- Each process gets a time slice to execute. If it doesn't finish within the time slice, it goes to the back of the queue.
- Advantages: Fair allocation; every process gets an equal share of CPU time.
- Disadvantages: Inefficient if the time slice is too large or too small (frequent context switches).
- Example:
  If processes 1, 2, and 3 have a time slice of 5ms:
  - Process 1 runs for 5ms, then Process 2 runs for 5ms, then Process 3.
  - If a process is incomplete, it is added back to the queue.

Summary

- Process Management ensures smooth execution and resource allocation for processes.
- Process Scheduling decides the execution order for processes.
- Preemptive Scheduling allows interruptions for higher-priority tasks, while Non-Preemptive Scheduling runs processes to completion.
- FCFS is simple but may lead to delays, while Round Robin ensures fairness with equal time allocation.

Memory Management and File System Management

1. Virtual Memory Techniques

- Definition: Virtual memory is a technique that allows a system to provide more memory resources than the physical RAM available by creating an illusion of larger memory.
- Main Concept:
    - When physical RAM is insufficient, the operating system uses a portion of the hard disk as temporary memory, called swap space.
    - Data not currently needed in RAM is moved to the swap space, and retrieved back into RAM when required.

Example:
If you run a heavy application like Photoshop and your RAM is full, the OS moves some less-used data from RAM to swap space and brings it back to RAM when needed.

---

2. Page Replacement Algorithm

- Purpose: When virtual memory usage exceeds the capacity of RAM, the system decides which pages (blocks of memory) to move between RAM and disk.
- Common Algorithms:
    - FIFO (First In, First Out): The first page loaded into RAM is the first to be swapped out. Simple but not always efficient.
    - LRU (Least Recently Used): The page least recently accessed is swapped out. This is more efficient as it prioritizes frequently accessed data.

Example:
If you're playing a game while multiple applications are open, the OS decides which app's data to move out of RAM to free up space for the game.

---

3. Segmentation and Paging

- Segmentation:
    - Divides memory into logical segments, each with a specific purpose (e.g., code, data, stack).

    - Segments can vary in size and are meaningful to the program structure.
    - Example: In a program, the code, data, and stack are stored in separate segments.
- Paging:
    - Divides memory into fixed-size blocks called pages, which are mapped to physical memory frames.
    - Reduces fragmentation and simplifies memory allocation.
    - Example: A large program is split into multiple fixed-size pages, making memory management more efficient.

---

4. File System Organization

- Definition: File systems organize and manage the storage and access of files on a disk.
- Structure:
    - Files: Collections of data stored with a name and location.
    - Directories: Containers that group related files together in a hierarchical structure.

Example:
If you create a file named notes.txt, it is stored in a directory like Documents. The directory provides the file's path and organizes it logically.

File System Organization and Linux Overview

---

1. Physical File System Organization Techniques (FAT/NTFS)

FAT (File Allocation Table)

- Description: An older, simple file system that organizes files using a table to store the starting and ending block addresses of files.
- Versions:
    - FAT12, FAT16, FAT32 (FAT32 is the most commonly used).

- Pros:
    - Lightweight and easy to implement.
- Cons:
    - Not efficient for managing large volumes.

Example: Storing small files on USB drives often uses FAT32 due to its simplicity.

NTFS (New Technology File System)

- Description: A modern, efficient, and secure file system widely used in Windows operating systems.
- Features:
    - Supports large files and volumes.
    - Provides journaling, which helps recover data after a system crash.
    - Better security and file size management.
- Pros:
    - Efficient and secure.
    - Handles large files effectively.
- Cons:
    - More complex and resource-intensive compared to FAT.

Example: Windows systems use NTFS to organize and manage files efficiently, especially for large or secure data.

---

2. Introduction to Linux

Overview

- Linux: An open-source, Unix-based operating system developed by Linus Torvalds in 1991.
- Key Features:

- Open Source: Source code is publicly available for modification and customization.
- Unix-like: Similar to the Unix operating system in design.
- Multi-user and Multitasking: Allows multiple users and tasks simultaneously.

---

3. Brief History, Evolution, Variants, and Installation Options

History

- Originated from Unix.
- Linux kernel was created by Linus Torvalds, forming the foundation for operating systems built on it.

Evolution

- Started as a basic kernel.
- Now widely used across servers, desktops, smartphones, and embedded systems.
- The most popular OS for servers globally.

Linux Variants (Distributions)

- Ubuntu: User-friendly, ideal for beginners.
- Debian: Focuses on stability.
- Red Hat Enterprise Linux (RHEL): Provides enterprise-level support.
- CentOS: A free alternative to RHEL.
- Fedora: Integrates the latest technologies.
- Arch Linux: Minimalistic, rolling release model for advanced users.

Installation Options

1. Direct Installation: Install Linux directly on your system's hard drive.
2. Virtual Machine: Use tools like VirtualBox or VMware to install Linux in a virtual environment.

3. WSL (Windows Subsystem for Linux): Run a Linux environment on Windows 10/11 systems.

Getting Acquainted with the Linux Environment

Common Linux Commands

1. File and Directory Management

- ls: List directory contents.
  Example: ls - Displays contents of the current directory.
- cp: Copy files or directories.
  Example: cp file1.txt file2.txt - Copies file1.txt to file2.txt.
- mv: Move or rename files/directories.
  Example: mv oldname.txt newname.txt - Renames oldname.txt to newname.txt.
- rm: Remove files or directories.
  Example: rm file.txt - Deletes file.txt.
- mkdir: Create a new directory.
  Example: mkdir newdir - Creates a directory named newdir.
- rmdir: Remove an empty directory.
  Example: rmdir emptydir - Deletes emptydir if it is empty.
- cd: Change the current directory.
  Example: cd /home/user - Navigates to /home/user.
- pwd: Print the current working directory.
  Example: pwd - Displays the full path of the current directory.

---

2. File Viewing and Manipulation

- cat: Display file contents.
  Example: cat file.txt - Displays the contents of file.txt.
- tac: Display file contents in reverse order.
  Example: tac file.txt - Displays file.txt in reverse order.

- more: View file contents page by page.
  Example: more file.txt - Displays file.txt one page at a time.
- head: Show the first few lines of a file.
  Example: head file.txt - Displays the first 10 lines of file.txt.
- tail: Show the last few lines of a file.
  Example: tail file.txt - Displays the last 10 lines of file.txt.
- sort: Sort lines in a text file.
  Example: sort file.txt - Sorts file.txt alphabetically.
- grep: Search for text using patterns.
  Example: grep "search_text" file.txt - Searches for "search_text" in file.txt.

---

3. File Information

- file: Determine the type of a file.
  Example: file file.txt - Displays the type of file.txt.
- wc: Count lines, words, and characters in a file.
  Example: wc file.txt - Outputs the count of lines, words, and characters in file.txt.
- diff: Compare two files line by line.
  Example: diff file1.txt file2.txt - Compares file1.txt and file2.txt.

---

4. File Search

- locate: Quickly find files by name.
  Example: locate file.txt - Searches for file.txt.
- find: Search for files in a directory hierarchy.
  Example: find /home/user -name "file.txt" - Searches for file.txt in /home/user.

---

5. File Compression and Archiving

- gzip / gunzip: Compress or decompress files.
  Example: gzip file.txt - Compresses file.txt.
  Example: gunzip file.txt.gz - Decompresses file.txt.gz.
- zip / unzip: Compress and decompress .zip files.
  Example: zip archive.zip file1.txt file2.txt - Compresses file1.txt and file2.txt.
  Example: unzip archive.zip - Decompresses archive.zip.
- tar: Archive files.
  Example: tar -cvf archive.tar file1.txt file2.txt - Archives file1.txt and file2.txt.

---

6. System Information

- date: Display the current date and time.
  Example: date - Outputs the current date and time.
- cal: Display a calendar.
  Example: cal - Shows the calendar for the current month.
- time: Measure the time taken by a command.
  Example: time ls - Measures the time taken to execute the ls command.

---

7. Command Help

- man: Display the manual for a command.
  Example: man ls - Shows the manual for the ls command.
- whatis: Get a short description of a command.
  Example: whatis ls - Displays a short description of ls.
- whereis: Locate the binary, source, and manual page of a command.
  Example: whereis ls - Displays locations for ls.

---

8. Links

- ln: Create hard links.
  Example: ln file1.txt link.txt - Creates a hard link to file1.txt named link.txt.

- ln -s: Create symbolic (soft) links.
  Example: ln -s file1.txt symlink.txt - Creates a symbolic link to file1.txt named symlink.txt.

touch – Create or Update a File

- Purpose: Create an empty file or update the timestamp of an existing file.
- Example:
  - touch newfile.txt - Creates an empty file named newfile.txt or updates its timestamp if it already exists.

---

2. echo – Display a Message

- Purpose: Print text or output a message to the terminal.
- Example:
  - echo "Hello, World!" - Prints "Hello, World!" to the terminal.

---

3. who – Display Logged-in Users

- Purpose: Show a list of users currently logged into the system.
- Example:
  - who - Displays a list of currently logged-in users.

---

4. finger – Show Detailed User Information

- Purpose: Display detailed information about a specific user.
- Example:
  - finger user1 - Displays information like the user's full name, login time, and other details about user1.

---

5. w – Display Logged-in Users and Their Activities

- Purpose: Show which users are logged in and what they are doing.

- Example:
    - w - Lists all logged-in users along with their current activities.

---

6. whoami – Show Current User

- Purpose: Display the username of the currently logged-in user.
- Example:
    - whoami - Prints the username of the current session.

---

7. alias and unalias – Create or Remove Aliases

- Purpose: Create shortcuts (aliases) for commands or remove them.
- Examples:
    - alias ll='ls -l' - Creates an alias ll for the ls -l command.
    - unalias ll - Removes the ll alias.

---

8. pushd and popd – Directory Stack Management

- Purpose: Save and restore the current directory path.
- Examples:
    - pushd /home/user - Saves the current directory path and switches to /home/user.
    - popd - Returns to the previously saved directory.

---

9. jobs – List Active Background Jobs

- Purpose: Display background jobs currently running in the shell.
- Example:
    - jobs - Lists all active background jobs in the terminal.

---

10. ps – Show Running Processes

- Purpose: Display a list of currently running processes.
- Example:
    - ps - Lists processes for the current shell session.

Summary of Key Commands

1. File Management:
    - ls, touch, cp, mv, rm, mkdir, rmdir.
2. File Viewing:
    - cat, tac, head, tail, more.
3. Search and Locate:
    - find, locate, grep.
4. System Information:
    - date, cal, w, who, whoami, ps.
5. File Compression:
    - gzip, gunzip, tar, zip, unzip.
6. Custom Shortcuts:
    - alias, unalias.
7. User and Session Management:
    - finger, jobs, pushd, popd.

Assignment – Lab: Getting Acquainted with the Linux Environment

The objective of this lab is to familiarize yourself with essential Linux commands for efficiently managing files, directories, and the system. Below are detailed explanations and practical examples of commonly used commands.

---

1. ls – List Directory Contents

- Purpose: Display the contents of the current or specified directory.
- Examples:

    - ls - Lists all files and folders in the current directory.
    - ls -l - Provides detailed information about files, including permissions, owner, size, and modification date.

---

2. cp – Copy Files or Directories

- Purpose: Copy files or directories to another location.
- Examples:
    - cp file1.txt file2.txt - Copies file1.txt to file2.txt.
    - cp -r dir1 dir2 - Recursively copies the dir1 directory and its contents to dir2.

---

3. mv – Move or Rename Files/Directories

- Purpose: Move files or directories to another location or rename them.
- Examples:
    - mv file1.txt /home/user/ - Moves file1.txt to the /home/user/ directory.
    - mv oldfile.txt newfile.txt - Renames oldfile.txt to newfile.txt.

---

4. rm – Remove Files or Directories

- Purpose: Delete files or directories.
- Examples:
    - rm file1.txt - Deletes the file file1.txt.
    - rm -r dir1 - Deletes the directory dir1 along with all its contents.

---

5. mkdir – Create a Directory

- Purpose: Create a new directory.
- Example:

    - mkdir newdir - Creates a directory named newdir.

---

6. rmdir – Remove an Empty Directory

- Purpose: Delete an empty directory.
- Example:
    - rmdir emptydir - Removes the directory emptydir if it is empty.

---

7. cd – Change Directory

- Purpose: Change the current working directory.
- Examples:
    - cd /home/user/ - Switches to the /home/user/ directory.
    - cd ~ - Navigates to the home directory.

---

8. pwd – Print Working Directory

- Purpose: Display the absolute path of the current directory.
- Example:
    - pwd - Prints the full path of the directory you are currently in.

9. cat – Display File Contents

- Purpose: View the contents of a file in the terminal.
- Example:
    - cat file1.txt - Displays the content of file1.txt.

---

10. grep – Search for Patterns in Files

- Purpose: Search for a specific string or pattern in a file.
- Example:
    - grep "search_text" file1.txt - Searches for search_text in file1.txt.

---

11. man – Manual Pages for Commands

- Purpose: Display the manual or documentation for a command.
- Example:
    - man ls - Opens the manual for the ls command.

---

12. echo – Print Text to the Terminal

- Purpose: Print custom text or variables in the terminal.
- Example:
    - echo "Hello, Linux!" - Prints "Hello, Linux!" on the terminal.

---

13. touch – Create Empty Files or Update Timestamps

- Purpose: Create a new empty file or update the timestamp of an existing file.
- Example:
    - touch newfile.txt - Creates an empty file named newfile.txt.

---

14. tar – Archive Files

- Purpose: Compress multiple files or directories into a single archive.
- Example:
    - tar -cvf archive.tar file1.txt file2.txt - Creates an archive archive.tar containing file1.txt and file2.txt.

---

15. zip and unzip – Compress and Decompress Files

- Purpose: Compress files using zip and decompress them using unzip.
- Examples:

- zip archive.zip file1.txt file2.txt - Compresses file1.txt and file2.txt into archive.zip.
- unzip archive.zip - Extracts files from archive.zip.

Task Assignment

Complete the following tasks to practice Linux commands:

1. Create a directory:
    - Command: mkdir Linux_lab
    - Creates a directory named Linux_lab.
2. Create three files inside the directory:
    - Command: touch file1.txt file2.txt file3.txt
    - Creates file1.txt, file2.txt, and file3.txt inside Linux_lab.
3. Copy a file:
    - Command: cp file1.txt file4.txt
    - Copies file1.txt to file4.txt.
4. Rename the copied file:
    - Command: mv file4.txt newfile.txt
    - Renames file4.txt to newfile.txt.
5. List directory contents:
    - Command: ls
    - Displays all files in the Linux_lab directory.
6. Display file content:
    - Command: cat file1.txt
    - Shows the content of file1.txt.
7. Search for text in a file:
    - Command: grep "word_to_search" file1.txt
    - Searches for word_to_search in file1.txt.

8. Compress files:
    - Command: tar -cvf files.tar file1.txt file2.txt file3.txt
    - Compresses the files into files.tar.
9. Remove files and directory:
    - Command:
        - rm file1.txt file2.txt file3.txt - Deletes the files.
        - rmdir Linux_lab - Deletes the empty directory.

Text Editors: vi and nano

vi Editor

- A powerful text editor with two main modes:
    - Command Mode: For issuing editing commands.
    - Insert Mode: For typing text.
- Examples:
    - Open a file: vi file.txt
    - Switch to Insert Mode: Press i.
    - Save and Exit:
        - Save: :w
        - Quit: :q
        - Save and Quit: :wq

---

nano Editor

- A simple, user-friendly text editor.
- Examples:
    - Open a file: nano file.txt
    - Save changes: Press Ctrl + O, then Enter.

    - Exit: Press Ctrl + X.

Linux File System

- Organized in a hierarchical tree structure starting from the root directory (/).
- Important Directories:
    - / - Root directory.
    - /bin - Contains essential command binaries.
    - /home - Stores user directories.
    - /etc - System configuration files.
    - /var - Logs and variable data files.
    - /usr - User programs and data.

Disk Partitioning

Partitioning divides a disk into logical sections for better management.

- Use fdisk for creating or modifying partitions.
- Example:
    - sudo fdisk /dev/sda - Opens the partitioning tool for the specified disk.

File Permissions

- Permission Types:
    - Read (r): Allows reading the file.
    - Write (w): Allows modifying the file.
    - Execute (x): Allows running the file as a program.
- Change Permissions:
    - Command: chmod 755 file.txt

    - Grants the owner full permissions (read, write, execute) and read/execute for others.

Process Management

- ps: View running processes.
    - Command: ps aux
- kill: Terminate a process.
    - Command: kill PID
- top: Real-time view of processes.
    - Command: top

Linux Boot Process

1. BIOS/UEFI: Initializes hardware.
2. Bootloader: Loads the operating system (e.g., GRUB).
3. Kernel: Initializes the system and interacts with hardware.
4. Init: Starts system processes and enables user logins.

Session 5 & 6: Working with Linux

1. Introduction to Editors: vi and nano

- vi Editor: A powerful text editor in Linux that works in a command-line interface with two main modes:
    - Command Mode: For executing commands.
    - Insert Mode: For typing text.
    - Common commands:
        - vi file.txt: Open file in vi.
        - Press i to enter Insert Mode, Esc to return to Command Mode.
        - To save, use :w. To quit, use :q. For both, use :wq.

- nano Editor: A simpler, more user-friendly editor.
    - Open a file with nano file.txt.
    - To save: Ctrl + O, then Enter.
    - To exit: Ctrl + X.

2. The Linux File System

Linux has a tree-structured file system:

- /bin: Basic command binaries.
- /home: User home directories.
- /etc: System configuration files.
- /var: Variable data files.
- /usr: User programs and data.

3. Disk Partitioning

Disk partitioning divides a hard drive into logical sections.

- Use fdisk to partition:
    - sudo fdisk /dev/sda
    - Commands within fdisk:
        - m: Display help.
        - n: Create a new partition.
        - d: Delete a partition.
        - w: Save changes.

4. Working with Files and Directories

- Create directories: mkdir dir_name.
- Remove files: rm file_name.
- Copy files: cp source_file destination_file.
- Move/rename files: mv old_name new_name.

5. File Permissions and Access Control

Linux file permissions are critical for security.

- Permissions: Read (r), Write (w), Execute (x).
    - chmod 755 file.txt changes file permissions.
- Access Control Lists (ACL): For fine-grained permission control.
    - setfacl -m u:username:rw file_name sets ACL.
    - getfacl file_name displays ACL.

6. Process-Related Commands: fork(), kill

- fork(): Creates a new process.
- kill: Terminates a process.
    - kill 1234: Terminates process ID 1234.
- ps aux: Lists all running processes.
- top: Displays system's top processes.

7. Linux Boot Process

The boot process involves several stages:

1. BIOS/UEFI: Initializes hardware.
2. Bootloader: Loads the OS (e.g., GRUB).
3. Kernel: Initializes system and interacts with hardware.
4. Init: Starts the system and user login.

8. Startup Files

Some important startup files executed during boot:

- /etc/rc.local: Executes custom commands at startup.
- /etc/inittab: Defines system initialization and runlevels.

9. Installation of Linux OS

Steps for installing Linux:

1. Boot from installation media.
2. Select language and keyboard layout.

3. Partition disk.
4. Select packages to install.
5. Set root password and user credentials.
6. Complete installation and reboot the system.

Session 7: Linux Service Management & Commands

1. Controlling and Managing Services

- systemctl command (preferred method):
    - systemctl start service_name: Starts a service.
    - systemctl stop service_name: Stops a service.
    - systemctl restart service_name: Restarts a service.
    - systemctl enable service_name: Enables service at boot.
    - systemctl status service_name: Checks service status.
- service command (older systems):
    - service service_name start: Starts a service.
    - service service_name stop: Stops a service.

2. Access Control List (ACL) and chmod

- ACL: Fine-grained file permissions.
    - setfacl -m u:username:rw file_name: Set ACL.
    - getfacl file_name: View ACL.
- chmod Command: Changes file permissions.
    - chmod 755 file_name: Sets specific read, write, and execute permissions.

3. chown and chgrp Commands

- chown: Changes the ownership of a file.
    - chown user1:group1 file_name: Changes owner and group.
    - chown user1 file_name: Changes only the owner.

- chgrp: Changes only the group of a file.
    - chgrp group1 file_name: Changes the group.

4. Network Commands

- telnet: Connects to remote systems (insecure, replace with ssh).
- ftp: Transfers files to/from remote systems.
- ssh: Securely connects to remote systems.
- sftp: Secure file transfer (encrypted).
- finger: Displays user information.

5. Overview of Log Management

Logs help track system operations and errors.

- Log Locations:
    - /var/log/syslog or /var/log/messages: System logs.
    - /var/log/auth.log: Authentication logs.
    - /var/log/daemon.log: Daemon logs.
- journalctl Command: Views systemd logs.
    - journalctl -u service_name: Show logs for a service.

Assignment Explanation

1. Command to Create a Directory, Create 10,000 Files, Set Permissions, and Remove Read Permissions for Others:

bash

CopyEdit

mkdir new_directory && cd new_directory && touch file{1..10000} && chmod 755 * && chmod o-r *

Explanation:

- Creates a directory, enters it, creates 10,000 files, sets file permissions to 755, and removes read permissions for others.

2. Set Default Permissions for Files and Directories: Use umask to set default permissions:

bash

CopyEdit

umask 022

Explanation:

- Files will have default permissions of 644 (rw-r--r--) and directories will have 755 (rwxr-xr-x).

System Configuration & Network Management

1. System Configuration Files

System configuration files define the behavior of the system and applications in Linux. These files are typically located in the /etc/ directory. Some important configuration files include:

- /etc/passwd: Stores details of all users in the system, including username, home directory, and login shell.
    - Command: cat /etc/passwd
- /etc/fstab: Specifies disk partitions and their mounting points.
    - Command: cat /etc/fstab
- /etc/network/interfaces: Configures network interfaces (used in older Debian-based systems).
    - Command: cat /etc/network/interfaces
- /etc/hostname: Stores the hostname of the system.
    - Command: cat /etc/hostname
- /etc/hosts: Maps hostnames to IP addresses.
    - Command: cat /etc/hosts

2. Network Configuration

To configure the network, you can set up IP addresses, network interfaces, and troubleshoot network-related issues.

- Setting IP Addresses (Static & Dynamic):
    - Static IP Configuration: Set a manual static IP by editing the /etc/network/interfaces file.
        - Example:

bash

CopyEdit

auto eth0

iface eth0 inet static

address 192.168.1.100

netmask 255.255.255.0

gateway 192.168.1.1

    - Dynamic IP Configuration (DHCP): Use DHCP to automatically obtain an IP address.
        - Example:

bash

CopyEdit

auto eth0

iface eth0 inet dhcp

- Bringing Network Interface Up/Down:
    - Use ifconfig or ip commands to bring network interfaces up or down.
        - Example:

bash

CopyEdit

ifconfig eth0 up   # Brings the interface up

ifconfig eth0 down # Brings the interface down

Or with ip:

bash

CopyEdit

ip link set eth0 up # Brings the interface up

ip link set eth0 down # Brings the interface down

3. Network Monitoring and Troubleshooting

Useful commands for monitoring and troubleshooting network interfaces and configurations:

- netstat Command: Displays network connections, routing tables, and interface statistics.
    - Example commands:
        - Show all active connections: netstat -a
        - Show listening ports: netstat -tuln
        - Show network interface statistics: netstat -i
- iproute2 Commands: A powerful suite for managing and troubleshooting network configurations.
    - View IP addresses: ip addr show
    - Display routing table: ip route show
    - Add a new route: ip route add 192.168.1.0/24 via 192.168.1.1
    - Delete a route: ip route del 192.168.1.0/24
- Other Troubleshooting Commands:
    - Ping: Checks network connectivity to a remote host.
        - Example: ping 192.168.1.1
    - Traceroute: Traces the network path that packets take to reach a destination.
        - Example: traceroute 192.168.1.1

4. Basic Network/Remote Access Commands

- Setting IP Address:

    - Assign an IP address using either ifconfig or ip commands.
        - Example with ifconfig:

bash

CopyEdit

ifconfig eth0 192.168.1.100 netmask 255.255.255.0

        - Example with ip:

bash

CopyEdit

ip addr add 192.168.1.100/24 dev eth0

- Ping Command: Tests network connectivity.
    - Example: ping google.com or ping 192.168.1.1
- SSH (Secure Shell): Securely access remote servers.
    - Example:
        - SSH to a remote server: ssh user@192.168.1.1
        - SSH with a specific port: ssh -p 2222 user@192.168.1.1

Session 9 & 10: Introduction to BASH CLI & Scripting

1. Introduction to BASH Command Line Interface (CLI)

BASH (Bourne Again Shell) is a powerful shell that provides a command-line interface (CLI) for Linux and UNIX systems. It is used for running commands, controlling the system, and scripting. The CLI is flexible and allows direct interaction with the system.

- Basic Syntax:
    - Commands are written in the CLI to execute tasks.
    - Example: ls -l or pwd

2. Shell Variables and User-defined Variables

- Shell Variables: These are predefined variables in the system that represent the state of the system, such as $HOME, $PATH, etc.

    - Example:
        - $HOME: User's home directory.
        - $USER: The current logged-in user.
- User-defined Variables: These are variables that you define in your script. To define a variable, use the = symbol without spaces.
    - Example:

bash

CopyEdit

myvar="Hello World"

echo $myvar # Output: Hello World

3. Command-Line Arguments

When running a Bash script, you can pass command-line arguments, which are represented as $1, $2, ..., $N.

- Example script:

bash

CopyEdit

#!/bin/bash

echo "First argument: $1"

echo "Second argument: $2"

If the script is run with ./script.sh arg1 arg2, the output will be:

sql

CopyEdit

First argument: arg1

Second argument: arg2

4. Expansions in BASH

Bash has several types of expansions that help in performing tasks efficiently:

- Pathname Expansion: Expands file paths using wildcards like * and ?.

    - Example: ls *.txt lists all .txt files.
- Tilde Expansion (~): Represents the user's home directory.
    - Example: cd ~ navigates to the home directory.
- Arithmetic Expansion: Used for performing mathematical calculations.
    - Example:

bash

CopyEdit

result=$((5 + 3))

echo $result # Output: 8

- Brace Expansion: Generates multiple items.
    - Example: echo {1..5} outputs: 1 2 3 4 5
- Parameter Expansion: Expands the value of a variable.
    - Example:

bash

CopyEdit

var="Hello"

echo ${var}world # Output: Helloworld

- Command Substitution: Assigns the output of a command to a variable.
    - Example:

bash

CopyEdit

current_date=$(date)

echo $current_date

5. Relational and Logical Operators

Relational and logical operators are used in conditions in Bash scripting:

- Relational Operators:

    - -eq: Equal to
    - -ne: Not equal to
    - -gt: Greater than
    - -lt: Less than
    - -ge: Greater than or equal to
    - -le: Less than or equal to

Example:

bash

CopyEdit

```
if [ $a -eq $b ]; then
  echo "a is equal to b"
fi
```

- Logical Operators:
    - &&: Logical AND
    - ||: Logical OR

Example:

bash

CopyEdit

```
if [ $a -eq $b ] && [ $c -gt $d ]; then
  echo "Both conditions are true"
fi
```

6. User Input and Output

- User Input: To take input from the user, use the read command.
    - Example:

bash

CopyEdit

```
echo "Enter your name:"
read name
echo "Hello, $name!"
```

- Output: To print output, use the echo command.
    - Example: echo "This is a message."

7. Arithmetic in BASH

For arithmetic operations, use (( )).

- Example:

bash

CopyEdit

```
a=5
b=3
((sum=a+b))
echo $sum  # Output: 8
```

- You can also evaluate complex arithmetic expressions. Example:

bash

CopyEdit

```
((result = (5 + 3) * 2))
echo $result  # Output: 16
```

8. Bash Calculator

Use the bc command for performing calculations in Bash.

- Example:

bash

CopyEdit

```
echo "5+3" | bc  # Output: 8
echo "scale=2; 5/3" | bc  # Output: 1.66
```

9. If, Nested if, and Case Statements

- If Statement: Used to check conditions.
    - Example:

bash

CopyEdit

```
if [ $a -gt $b ]; then
  echo "a is greater than b"
fi
```

- Nested If: Used when you need to check another condition inside an existing if statement.
    - Example:

bash

CopyEdit

```
if [ $a -gt $b ]; then
  if [ $a -lt $c ]; then
    echo "a is between b and c"
  fi
fi
```

- Case Statement: Used for multiple conditions.
    - Example:

bash

CopyEdit

```
case $day in
  Monday)
    echo "Start of the week"
    ;;
  Tuesday)
```

```
  echo "Second day"
  ;;
 *)
  echo "Other day"
  ;;
esac
```

10. Loops: for, while, break, continue

- For Loop: Used for a fixed number of iterations.
    - Example:

```
bash
CopyEdit
for i in {1..5}; do
  echo "Iteration $i"
done
```

1. While Loop:

The while loop continues to execute as long as the condition is true.

- Example:

```
bash
CopyEdit
count=1
while [ $count -le 5 ]; do
  echo "Count is $count"
  ((count++))
done
```

2. Break Statement:

The break statement is used to terminate the loop.

- Example:

bash

CopyEdit

```
for i in {1..10}; do
  if [ $i -eq 5 ]; then
    break
  fi
  echo "i is $i"
done
```

3. Continue Statement:

The continue statement is used to skip the current iteration of a loop.

- Example:

bash

CopyEdit

```
for i in {1..5}; do
  if [ $i -eq 3 ]; then
    continue
  fi
  echo "i is $i"
done
```

4. Variable & String Manipulation:

Bash allows several string operations such as substring extraction, concatenation, and length calculation.

- Examples:
    - Substring: ${string:position:length}
    - Concatenation:

bash

CopyEdit

```
string1="Hello "
string2="World"
echo $string1$string2
```

- Length: ${#string}

Summary of Sessions 9 & 10:

- We covered the BASH CLI and learned about shell variables, user-defined variables, command-line arguments, expansions, operators, loops, and conditions.
- These concepts are fundamental for efficient and flexible scripting in BASH.

Assignment – Lab:

You will write shell scripts that display system information and perform operations. Here's a breakdown of tasks:

1. Shell Script to Return System Information:

This script fetches the system information such as hostname, logged-in users, disk space, memory usage, and system uptime.

- Script:

bash

CopyEdit

```
#!/bin/bash
echo "Hostname: $(hostname)"
echo "Logged-in users:"
who
echo "Disk space usage:"
df -h
```

```
echo "Memory usage:"
free -h
echo "System uptime and Load:"
uptime
```

- Explanation:
  - hostname: Displays the system's hostname.
  - who: Shows logged-in users.
  - df -h: Displays disk space usage in a human-readable format.
  - free -h: Shows memory usage in a human-readable format.
  - uptime: Displays system uptime and load averages.

2. Shell Script to Add .new Extension to All Files in a Directory:

This script traverses a directory and appends the .new extension to all files.

- Script:

```
bash
CopyEdit
#!/bin/bash
echo "Enter the directory path:"
read dir
for file in "$dir"/*; do
  if [ -f "$file" ]; then
    mv "$file" "$file.new"
    echo "Renamed $file to $file.new"
  fi
done
```

- Explanation:
  - read dir: Takes directory path input from the user.

- for file in "$dir"/*: Iterates through all files in the directory.
- mv "$file" "$file.new": Adds .new extension to each file.

3. Shell Script to Perform Addition or Subtraction Based on Arguments:

This script performs either addition or subtraction based on the user's command-line input.

- Script:

bash

CopyEdit

```
#!/bin/bash
if [ $# -lt 3 ]; then
  echo "Usage: $0 <num1> <operation (+ or -)> <num2>"
  exit 1
fi
num1=$1
operation=$2
num2=$3
if [ "$operation" == "+" ]; then
  result=$((num1 + num2))
  echo "Result: $num1 + $num2 = $result"
elif [ "$operation" == "-" ]; then
  result=$((num1 - num2))
  echo "Result: $num1 - $num2 = $result"
else
  echo "Invalid operation. Use + or -."
  exit 1
fi
```

- Explanation:

- if [$# -lt 3]: Checks if there are fewer than 3 arguments, shows usage if true.
- num1=$1, operation=$2, num2=$3: Stores user input in variables.
- Checks for + or - operations and performs the corresponding arithmetic.

Session 11 & 12: Bash Scripting Concepts

1. Search: grep and find

- grep (Global Regular Expression Print):
  - Purpose: grep is used to search for a specific pattern or string within a text.
  - Syntax: grep "pattern" filename
    - "pattern" represents the search string, and filename is the file in which you want to search.
-
  - Example: grep "error" log.txt - This will display all lines in log.txt that contain the word "error".
- find: Used to search for specific files in a directory based on conditions like filename, size, or type.
  - Syntax: find /path/to/directory -name "filename"
  - Example: find /home/user -name "*.txt" - This will search for all .txt files in the /home/user directory.

2. Error Handling

- Exit Status: Every command returns an exit status. 0 means successful execution, and a non-zero value indicates an error.
  - Example:

bash

CopyEdit

#!/bin/bash

cp file1.txt /home/user/ || { echo "Copy failed"; exit 1; }

    - This script will display "Copy failed" if the cp command fails and exit with status 1.

- Trap Command: Used to catch specific signals like interruption (Ctrl+C) and take action.
    - Example:

bash

CopyEdit

#!/bin/bash

trap "echo 'Script interrupted!'; exit" SIGINT

3. Debugging & Redirection

- Debugging Commands:
    - set -x: Traces the execution of each command.
    - set -e: Stops the script if any command returns a non-zero status.
- Redirection:
    - Output Redirection (> and >>):
        - > overwrites the content of a file.
        - >> appends output to a file.
        - Example: echo "Hello, world!" > file.txt will overwrite file.txt with "Hello, world!".
        - Example: echo "New line" >> file.txt will append "New line" to file.txt.
    - Input Redirection (<): Directs the content of a file into a command.
        - Example: sort < file.txt will sort the content of file.txt.

4. Conditional Statements in Bash

- If Statement:
    - Syntax:

bash

CopyEdit

```
if [ condition ]; then
  # commands
fi
```

- If-Else Statement:
    - Syntax:

bash

CopyEdit

```
if [ condition ]; then
  # commands if true
else
  # commands if false
fi
```

- Nested If: An if inside another if.
    - Example:

bash

CopyEdit

```
if [ condition1 ]; then
  if [ condition2 ]; then
    # commands
  fi
fi
```

- Case Statement: Used for multiple conditions.
    - Syntax:

bash

CopyEdit

```
case $variable in
  pattern1)
    # commands ;;
  pattern2)
    # commands ;;
  *)
    # default case ;;
esac
```

5. Automating Tasks Using Bash Scripts

- Example: Backup Script:
    - Automate backing up important files:

bash

CopyEdit

```
#!/bin/bash
SOURCE_DIR="/home/user/data"
BACKUP_DIR="/home/user/backups"
DATE=$(date +%F)
tar -czf $BACKUP_DIR/backup_$DATE.tar.gz $SOURCE_DIR
echo "Backup completed successfully on $DATE"
```

6. Security Patches

- Example: Apply Security Updates:
    - Automates applying security updates to the system:

bash

CopyEdit

```
#!/bin/bash
```

```
echo "Updating the system..."
sudo apt update && sudo apt upgrade -y
sudo apt install unattended-upgrades -y
sudo dpkg-reconfigure -plow unattended-upgrades
echo "System updated and security patches applied successfully."
```

7. Logging & Monitoring Using Bash Scripts

- Example: System Monitoring Script:
    - Logs system health metrics like disk usage, memory, processes, and uptime.

```
bash
CopyEdit
#!/bin/bash
LOGFILE="/var/log/sysmonitor.log"
echo "System Monitoring Report - $(date)" >> $LOGFILE
echo "---------------------------------------" >> $LOGFILE
df -h >> $LOGFILE
free -h >> $LOGFILE
top -n 1 | head -n 10 >> $LOGFILE
uptime >> $LOGFILE
echo "---------------------------------------" >> $LOGFILE
```

8. Hands-on Linux Commands

- Basic Commands: ls, cp, mv, rm, mkdir, rmdir, pwd, cd, echo, cat, grep, ps, top, kill, chmod, chown.
    - Example: ps aux displays the status of running processes.

9. Vim Editor

- Basic Vim Commands:
    - i: Enter insert mode.

    - Esc: Return to normal mode.
    - :w: Save the file.
    - :q: Quit Vim.

10. User Management

- Adding a user: sudo useradd username, sudo passwd username.
- Deleting a user: sudo userdel username.
- Modifying a user: sudo usermod -aG groupname username.

11. Working with Processes (ps, top, kill)

- ps: View running processes.
- top: Real-time process monitor.
- kill: Terminate a specific process.

12. Cron Jobs

- Used for automating repetitive tasks at scheduled times (e.g., backups, updates).

13. Error and Exception Handling in Programs

- Error handling: Use try-catch blocks to manage exceptions in scripts.

14. Implementing OOP Concepts in Perl

- Use object-oriented programming concepts like classes, inheritance, and polymorphism in Perl.

15. Working with MySQL and HTML Forms

- MySQL: Store data in a database.
- HTML Forms: Collect user data and store it in MySQL.

Conclusion:

- Hands-on experience with Linux commands and bash scripting is crucial for system administration.
- Automating tasks, monitoring system health, and handling security updates are common uses of bash scripting.

- Using commands like sudo, chown, and chmod enhances your system administration skills.
- Understanding MySQL and HTML form handling prepares you for web-based applications development.

4o mini