Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)

MoE, UGC & AICTE Approved

NAAC A++ Accredited

Department of Computer Science and Engineering

School of Computer Science and Technology

Lab Manual

23CS2047 Unix and Linux Lab (0:0:3)

Karunya INSTITUTE OF TECHNOLOGY AND SCIENCES

(Declared as Deemed to be University under Sec.3 of the UGC Act, 1956)

MoE, UGC & AICTE Approved

NAAC A++ Accredited

Department of Computer Science and Engineering

School of Computer Science and Technology

Lab Manual

23CS2047 Unix and Linux Lab (0:0:3)

Prepared by	Verified by	Approved by
Dr. I. Titus		

INDEX

VISION

To raise world-class Computer Science and Engineering professionals excelling in academics, research, and providing solutions to human problems.

MISSION

1. To develop professionals with strong fundamental concepts, programming, and problem-solving skills with exposure to emerging technologies.

2. To promote research in the state of art technologies, providing solutions to human problems, especially in the areas of Health, Water, Energy, and Food.

3. To develop leadership qualities with ethical values to serve society.

Program Educational Objectives (PEOs)

Graduates will

- **PEO I:** Demonstrate the knowledge acquired to design and develop innovative software solutions.
- **PEO II:** Exhibit technical skills and excel as computer professionals, academicians, researchers, and entrepreneurs.
- **PEO III:** Execute professional practice to serve society with ethics.

Program Outcomes (POs)

Graduates will have the ability to:

- **PO1-Engineering Knowledge:** Apply the knowledge of mathematics, natural sciences, and engineering fundamentals specialization to the solution of complex engineering problems.
- **PO2-Problem Analysis:** Identify, formulate, review research literature, and analyze complex Engineering Problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences.
- **PO3-Design / Development of Solutions:** Design solutions for complex engineering problems and design system components or processes that meet specific needs with appropriate consideration for the public health and safety and the cultural, societal and environmental considerations.
- **PO4-Conduct Investigations of Complex Problems:** Use research- based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of the information to provide valid conclusions.
- **PO5-Modern Tool Usage:** Create, Select and Apply appropriate techniques, resources and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
- **PO6-The Engineer and Society:** Apply Reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7-Environment and Sustainability:** Understand the impact of the professional

engineering solutions in societal and environmental contexts and demonstrate the knowledge of and need for sustainable development.

- **PO8-Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9-Individual and Team Work:** Function effectively as an individual and as a member or leader in diverse teams and in multi-disciplinary settings.
- **PO10-Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large such as being able to comprehend and write effective reports and design documentation make effective presentations and give and receive clear instructions.
- **PO11-Project Management and Finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply this to one’s own work, as a member and leader in a team, to manage projects and in multi-disciplinary environments.
- **PO12-Life Long Learning:** Recognize the need for and have the preparation and ability to engage in independent and lifelong leaning in the broadest context of technological change.

Program Specific Outcome (PSOs)

Graduates will have the ability to

- **PSO 1:** Understand, analyse and develop software products and solutions using standard practices and strategies in the areas related to algorithms, system software, multimedia, web design, database, and networking for effective and efficient design of computer-based systems of varying complexity.
- **PSO 2:** Employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies

SYLLABUS

Course Code	Unix and Linux Lab	L	T	P	C
23CS2047		**0**	**0**	**3**	**1.5**

Course Objectives:

Enable the students to:

1. Describe Unix and Linux system administration commands.
2. Illustrate file and process creation and management in operating systems.
3. Apply advanced system concepts such as inter - process communication, and networking.

Course Outcomes:

The students will be able to:

1. Examine advanced Unix and Linux commands used for system administration.
2. Construct shell scripts to automate various system administration tasks.
3. Develop expertise in managing files and directories.
4. Create and manage processes effectively using system calls.
5. Simulate various inter - process communication and synchronization techniques like pipes, semaphores, shared memory.
6. Write networking and firewall configuration rules to protect against unauthorized access and malicious activities.

Software Required if any:

Virtual Box - Linux OS

List of Exercises

1	Basic Unix and Linux Commands
2	Advanced Unix Shell Commands
3	Basic Shell Programming
4	User defined functions using shell scripting
5	File processing and management
6	Process Creation and Management using system calls
7	Process Scheduling
8	Inter - process Communication using Pipes
9	Inter - process Communication using Semaphores and Shared memory
10	Memory and Disk Management
11	Network configuration - IP addressing
12	Firewall Configuration

Theme Details

Note: Each student will complete all exercises based on their assigned theme to develop unique, scenario-based solutions. The theme includes, but is not limited to, the following

1. **Digital Document Library Management**
2. **Personal Finance Tracker**
3. **Movie Rental System**
4. **Online Store Inventory System**
5. **Employee Time Tracking System**
6. **Student Academic Records System**
7. **Hotel Reservation System**
8. **Restaurant Order Management System**
9. **Social Media Profile Management**
10. **Online Voting System**
11. **Digital Photo Album Organizer**
12. **Travel Itinerary Manager**
13. **Grocery Shopping List Management**
14. **Event Ticket Booking System**
15. **Subscription Management System**
16. **Library Catalogue and Loan System**
17. **Smart Home Automation Logs**
18. **Fitness Tracker Data Management**
19. **E-commerce Transaction Log System**
20. **Job Application Tracking System**
21. **Research Paper Citation Manager**
22. **Online Forum User Management**
23. **Movie Database Management**
24. **Task Scheduling and Reminders System**
25. **Server Logs and Monitoring System**
26. **Backup and Restore System**
27. **Data Encryption and Security Management**
28. **File Sharing and Distribution System**
29. **Student Exam Results System**
30. **Inventory and Stock Control System**
31. **Shipping and Delivery Management System**
32. **Budget Planning and Expense Tracker**
33. **Online Banking Transaction Records**
34. **Fleet Vehicle Tracking System**
35. **Cloud Storage Management System**
36. **Document Version Control System**
37. **Online Course Enrollment System**
38. **Health and Medical Records Management**
39. **Customer Support Ticket System**
40. **Rental Equipment Management System**
41. **IoT Device Data Logger**
42. **Weather Data Collection and Management**
43. **University Research Data System**
44. **Pet Adoption and Record Management**
45. **Construction Project Management System**
46. **Server Resource Usage Monitoring**
47. **Network Security Log Management**
48. **Personal Workout Log System**
49. **Online Game Score Tracking System**
50. **Data Cleansing and Transformation System**
51. **Customer Feedback System**
52. **Membership Database System**
53. **Contact Management System**
54. **Online Learning Progress Tracker**
55. **Wedding Planning System**
56. **Sales and Commission Tracking System**
57. **Project Task Management System**
58. **Movie Streaming Subscription System**
59. **Customer Loyalty Points System**
60. **Multi-tenant Cloud Resource System**
61. **Online Survey and Feedback System**
62. **File Compression and Backup System**
63. **Public Transportation Timetable System**
64. **Health Insurance Claim Tracker**
65. **Employee Payroll System**
66. **Smart Parking Management System**
67. **Auction Management System**
68. **Employee Skillset Database**
69. **Security Camera Log Management**
70. **Dynamic Pricing System for E-commerce**

Ex.no. 1 Exploring Unix Basics: Mastering File and Directory Management

Objective:
The objective is to familiarize students with the basic Unix and Linux commands for file and directory management, text processing, and system utilities. By performing hands-on tasks, students will:

- Learn to navigate the file system using essential commands.
- Understand how to create, view, and manage files and directories.
- Gain proficiency in searching and processing file content.
- Explore basic file permissions and system utilities.
- Develop foundational skills required for advanced Unix/Linux operations.

Job Roles: System Administrator, IT Support Specialist, Linux Administrator

Skills: File management, directory navigation, user account management

Prerequisites

1. **Familiarity with Basic Operating System Concepts**:
   - Understanding of file systems, directories, and file paths (absolute and relative).
   - Knowledge of user roles and permissions in an operating system.
2. **Command-Line Basics**:
   - Ability to access and use a terminal or command-line interface.
   - Awareness of how to type and execute commands.
3. **Understanding of Text Editors**:
   - Basic knowledge of how to use text editors like Vim, Nano, or gedit for creating and editing files.
4. **Access to a Unix/Linux Environment**:
   - A system with Unix/Linux installed or access to a virtual machine or cloud environment with a terminal.

Common Commands and Code Snippets:

List Contents of Directory

```
ls -a /path/to/your/theme/directory
```

Lists all files, including hidden files (those starting with a dot).

Create Directories

```
mkdir dir1 dir2
```

Creates two directories named dir1 and dir2.

Change Directory

```
cd dir1
```

Changes the current directory to dir1.

Copy a File

```
cp /etc/passwd sample.txt
```

Copies the /etc/passwd file to sample.txt in the current directory.

Display Working Directory

```
pwd
```

Prints the current working directory.

Display File Content

```
cat test1.txt
```

Displays the content of test1.txt on the terminal.

Remove a Directory

```
rmdir dir1
```

Removes the empty directory dir1.

Theme Based Questions

Basic Commands and File Management using the theme assigned.

1. List the contents of a specific directory, including hidden files.
2. List the contents of a system directory of your choice (e.g., /var, /tmp, etc.).
3. Create two directories with unique names (e.g., dirA and dirB).
4. Create a hidden directory with a custom name.
5. Change into one of the directories you just created.

File Operations

6. Copy a specific system file (e.g., /etc/passwd) to your current directory with a new name.
7. Change back to your home directory and verify your location using a command.
8. Create a file using a text editor (e.g., Vim, Nano) and add the following sample data:
    - Include custom rows of data relevant to your theme (e.g., user details, employee data, or product details).

File Content Analysis

9. Answer the following based on the file you created:
   a) Filter and display rows based on a specific condition (e.g., users with a specific interest).
   b) Display selected columns of the data (e.g., names and IDs).
   c) Count the rows matching a specific condition and save the result in a new file.
   d) Extract specific rows (e.g., first and last two) and save them into another file.
10. Display the contents of the created file without blank lines.

File and Directory Manipulation

11. Move a file from one directory to another.
12. Change into the target directory and check if the file is present there.
13. Rename a file in the target directory and verify the changes.

System Utilities

14. Display the calendar of a specific month and year.
15. Remove one of the directories you created earlier.

File Content Display

16. Display the last few lines (e.g., last 3) of a file.

Command History

17. List all the commands you have executed so far and save them into a file with a specific name (e.g., command_history.txt).

Counting and Sorting

18. Count the total number of files in a specific directory.
19. Perform sorting of the contents of three files and store the result in a fourth file.

File Permissions

20. Change the permissions of a file such that group users and others have no access.

Ex.no. 2:
The Art of Shell Mastery: Unleashing Advanced Unix Command Power

Objective: Use advanced commands to perform complex operations.

Job Roles : Unix/Linux Administrator, DevOps Engineer, IT Operations Specialist

Skills : Command-line proficiency, system monitoring, advanced file operations

Prerequisites

1. **Basic Knowledge of Unix Commands**:
   - Familiarity with basic commands for file and directory management, such as ls, cd, mkdir, rm, and cp.
   - Understanding of file paths, including absolute and relative paths.
2. **Command-Line Proficiency**:
   - Ability to navigate and work efficiently in a terminal environment.
   - Awareness of how to execute commands with options/flags (e.g., ls -l, grep -i).
3. **Understanding of File Permissions**:
   - Basic knowledge of Unix file permission structure (read, write, execute) and how to interpret it.
   - Awareness of user, group, and others in the context of permissions.
4. **Awareness of Process and System Management**:
   - Basic understanding of processes, their lifecycle, and system monitoring.
5. **Introduction to Text Processing**:
   - Familiarity with text files and basic file viewing commands (cat, less, more).
6. **Networking Basics (Optional)**:
   - General understanding of networking concepts, such as IP addresses and server connectivity, to perform related tasks.
7. **Access to a Unix/Linux Environment**:
   - A Unix/Linux system, virtual machine, or cloud environment with necessary tools and commands available.

Common Commands and Code Snippets:

Search Files by Name

```
find /path/to/your/theme/directory -name "*.txt"
```

Searches for all .txt files in the specified directory.

Count the Number of Files

```
find . -type f | wc -l
```

Counts the total number of files in the current directory and subdirectories.

Change File Permissions

```
chmod 755 test.txt
```

Changes the permissions of test.txt to rwxr-xr-x.

Search Within Files

```
grep "GridComputing" test1.txt
```

Searches for the term "GridComputing" within the file test1.txt.

List Detailed File Information

```
ls -l /path/to/your/theme/directory
```

Lists all files in long format, showing permissions, owner, group, size, and modification date.

Theme Based Questions

1. **System Navigation and File Operations**:
    - How can you list all files in a directory along with their details, such as size, permissions, and modification date?
    - Create a directory structure with nested subdirectories and files. Display the hierarchy using a single command.
2. **File and Directory Search**:
    - Find all files in a specific directory containing a particular string in their name.
    - Search for all files larger than a specified size within a given directory.
3. **Text Processing**:
    - Extract and display lines from a file that contain a specific keyword.
    - Count the number of occurrences of a specific word in a file.
    - Replace all occurrences of one string with another in a file and save the changes to a new file.
4. **Sorting and Filtering**:
    - Sort the contents of a file alphabetically and save the result to another file.
    - Display only unique lines from a file.
5. **File Compression and Archiving**:
    - Compress a set of files into a .tar.gz archive and list its contents.
    - Extract specific files from an existing archive.
6. **Process Management**:
    - Display all currently running processes on the system and identify the process consuming the most CPU.
    - Terminate a specific process by its process ID (PID).
7. **User and Group Management**:
    - List all currently logged-in users and display their active sessions.
    - Create a new user and assign them to a specific group.
8. **Permissions and Ownership**:
    - Change the permissions of a file so that only the owner has read, write, and execute access.
    - Modify the ownership of a file to a different user.
9. **System Monitoring and Utilities**:
    - Display the disk usage of all mounted file systems.
    - Monitor real-time memory usage on the system.
10. **Networking**:
    - Display the IP address of the system.
    - Check if a specific website or server is reachable using a command.
11. **Command History and Shortcuts**:
    - Display all commands executed during the session and save them to a file.
    - Create an alias for a commonly used command and demonstrate its usage.
12. **System Date and Time**:
    - Display the system's current date and time.
    - Schedule a command to run at a specific time using cron.

Ex.no. 3: Building Dynamic Automation: Foundations of Shell Programming

Objective: The objective is to learn basic shell programming skills, including writing and executing shell scripts for automating tasks, managing files, and processing text. Students will learn to use control structures like loops and conditionals, handle input/output, and perform system monitoring. The exercise aims to develop proficiency in writing scripts for system management and task automation in Unix/Linux environments.

Job Roles : Automation Engineer, SRE (Site Reliability Engineer), Build and Release Engineer

Skills : Task automation, script development, efficiency optimization

Prerequisites

To effectively perform Exercise 3, students should have the following foundational knowledge and skills:

1. **Basic Understanding of Unix/Linux Commands**:
   - Familiarity with basic commands for navigating directories, listing files (ls), changing directories (cd), and creating/removing files and directories (mkdir, rmdir, rm, touch).
   - Knowledge of how to view the contents of a file using commands like cat, more, or less.
2. **File Permissions and Ownership**:
   - Understanding how Unix file permissions work (r, w, x) and the concepts of file ownership (user, group, others).
   - Basic commands for changing permissions and ownership (chmod, chown).
3. **Understanding of the Shell Environment**:
   - Awareness of the Unix/Linux command-line interface and its structure.
   - Ability to use the shell to execute commands, chain commands using operators like &&, ||, and ;.
4. **Basic Text Manipulation**:
   - Familiarity with simple text processing commands (grep, sed, awk, cut, sort, uniq) for searching, filtering, and manipulating text data.
5. **Input and Output Redirection**:
   - Understanding the use of output redirection (>, >>, 2>) and input redirection (<).
   - Ability to pipe (|) output from one command to another.
6. **Basic Scripting Concepts**:
   - Knowledge of how to create and execute a basic shell script.
   - Understanding the structure of a shell script, including the use of comments (#), and executing scripts (./script.sh).
7. **Variables and Environment Variables**:
   - Basic knowledge of shell variables and how to use them in scripts.
   - Familiarity with environment variables and how to view or modify them (e.g., echo $HOME, export VAR=value).
8. **Control Structures (Conditional Statements)**:
   - Understanding of basic conditional statements (if, else, elif) to execute code based on conditions.
   - Knowledge of comparison operators (-eq, -ne, -lt, -gt, -le, -ge, -z, -n).
9. **Loops**:

    - Familiarity with for, while, and until loops for iteration over lists, files, or conditions.
10. **Basic Error Handling**:
    - Ability to check for errors in shell commands and handle them using exit status and logical operators.
11. **System Monitoring Commands**:
    - Understanding of basic system commands to display system information such as df, top, ps, free, uptime, who, etc.

<u>Common Commands and Code Snippets:</u>

Create a Simple Shell Script

```
#!/bin/bash
echo "Welcome to the theme directory!"
```

A simple script that prints a welcome message.

Input from User

```
read -p "Enter the book name: " bookname
echo "You entered: $bookname"
```

Prompts the user to input a book name and displays it.

If-Else Statement

```
if [ -f "test1.txt" ]; then
   echo "File exists"
else
   echo "File does not exist"
fi
```

Checks if test1.txt exists and prints a message based on that.

For Loop

```
for i in {1..5}
do
   echo "Processing file $i"
done
```

Loops from 1 to 5, printing a message each time.

<u>Theme Based Questions</u>

File and Directory Management:

- Write a script to list all files in a specified directory and display the total number of files.
- Write a script to check if a specified file or directory exists. If it exists, display its size or content; otherwise, display an error message.
- Develop a script to create a new directory and move a specified file into it.
- Create a script to rename a file or directory based on user input.

 File Processing:

- Write a script to search for a specific word or pattern in a file and display the line numbers where it occurs.
- Write a script to count the number of lines, words, and characters in a file.

- Create a script that reads a file, processes its content, and saves the result to another file (e.g., remove blank lines, replace certain text).

 User Interaction:

- Write a script that prompts the user to input their name, and then displays a personalized greeting.
- Develop a script that asks the user for an action (e.g., "Display system info", "List files") and executes the corresponding command.
- Create a script that takes two numerical inputs from the user and calculates their sum, difference, product, and quotient.

 Conditional Statements and Logic:

- Write a script that checks whether a specific directory is empty or not, and display an appropriate message.
- Create a script that checks if a file is readable or writable by the current user and displays the result.
- Develop a script that checks if a given file exists and is older than a certain number of days, and then archives it.

 Loops and Iterations:

- Write a script that prints all the numbers from 1 to 100.
- Create a script that calculates the factorial of a number using a loop.
- Write a script that iterates over all files in a directory and displays their names and permissions.

 Text and String Manipulation:

- Write a script to reverse a string entered by the user.
- Create a script that converts all characters in a string to uppercase or lowercase.
- Write a script to check if a string is a palindrome (reads the same forward and backward).

 File Compression and Archiving:

- Write a script to compress a directory into a .tar.gz archive and display the size of the archive.
- Create a script to extract files from a .tar archive and list the extracted files.

 System Monitoring and Process Management:

- Write a script to display the system's CPU usage, memory usage, and disk space.
- Create a script that lists all running processes and allows the user to terminate a specific process by its PID.
- Develop a script that displays system uptime and the number of users currently logged in.

 Permissions and Ownership:

- Write a script to change the permissions of all files in a directory to ensure only the owner has read and write access.

- Create a script that changes the owner and group of a specified file or directory.

 Automation and Scheduling:

- Write a script that performs regular backups of a specified directory and stores them in a backup directory.
- Create a script that schedules a task (e.g., system cleanup or file backup) to run at a specific time using cron jobs.

 Error Handling:

- Write a script that checks if a command or process has executed successfully and displays an appropriate success or error message.
- Create a script to handle errors in file operations (e.g., file not found or insufficient permissions) and display helpful error messages.

 Environment Variables and System Info:

- Write a script to display the system's current environment variables, such as PATH, HOME, and USER.
- Develop a script that sets a new environment variable and displays its value.

Ex.no. 4: Custom Function Frameworks: Elevating Shell Scripting Efficiency

Objective: The objective of this exercise is to understand how to create and use functions in shell scripting. Students will learn to define reusable blocks of code as functions, call these functions, and pass arguments to them. This exercise helps in writing modular and structured shell scripts.

Job Roles : Software Developer, Application Support Engineer, DevOps Engineer

Skills : Modular programming, reusable code creation, debugging shell scripts

Prerequisites:

Before performing this exercise, students should:

1. Understand the basic structure of shell scripts.
2. Be familiar with variables and conditional statements.
3. Know how to execute shell scripts.
4. Understand input and output redirection.

Common Commands and Code Snippets:

Define a function:

Functions in shell scripting are defined using the following syntax:

```
function function_name() {
    # Function body
}
```

Alternatively:

```
function_name() {
    # Function body
}
```

Call a function:

Functions are invoked by simply using their name:

```
function_name
```

Pass arguments to a function:

Function arguments are accessed using positional parameters (`$1`, `$2`, ...):

```
function_name arg1 arg2
```

Return values from a function:

Functions can return a value using the `return` keyword (limited to integer values) or by using `echo` to output a value:

```
return 0
```

Simple Function Definition

```
greet() {
   echo "Hello, $1!"
}
greet "Student"
```

Defines a function greet() that takes one argument and prints a greeting.

Function for File Copy

```
copy_file() {
   cp $1 $2
   echo "File $1 copied to $2"
}
copy_file "test1.txt" "/path/to/backup/"
```

A function to copy a file from source to destination.

Function to Create Directory

```
create_directory() {
   mkdir $1
   echo "Directory $1 created."
}
create_directory "dir3"
```

A function to create a directory with a given name.

Function with Conditional Logic

```
check_file() {
   if [ -f $1 ]; then
      echo "$1 exists."
   else
      echo "$1 does not exist."
   fi
}
check_file "test1.txt"
```

A function that checks if a file exists.

Function with Loop

```
list_files() {
   for file in $1/*; do
```

```
        echo "File: $file"
    done
}
list_files "/path/to/your/theme"
```

A function to list all files in the specified directory.

Theme Based Questions

1. Write a shell function to create a directory structure for your theme and display a message when it's created.
2. Develop a function to list all files related to your theme in a specified directory and categorize them based on type.
3. Create a shell function to accept user input for performing different tasks related to your theme, such as adding, deleting, or modifying records.
4. Implement a function to backup important theme-related files and log the success or failure of the operation.
5. Write a function that accepts two parameters: a source file and a destination. The function should copy the file to the destination and check if the copy operation was successful.
6. Develop a function that generates a report on the usage of a particular theme-related resource, like books in a library or items in a store.
7. Create a shell function that checks the status of theme-related services (like availability of books in a library system) and prints a message based on the availability.
8. Write a function to search for a specific theme-related keyword (e.g., a book title or product) in a file and return its position.
9. Develop a function to calculate the total number of theme-related items (e.g., total books or inventory) in a given directory.
10. Implement a function to handle user authentication for theme-related operations (e.g., borrowing a book, placing an order) and display appropriate access messages.

Ex.no. 5: File Dominance: Advanced Operations and Mastery in Unix Systems

Objective: The objective is to automate file processing tasks using shell scripting. This includes file manipulate, search, and filter file contents, manage file permissions, and handle file backups.

Job Roles : Data Engineer, Backup and Storage Specialist, System Analyst

Skills : File management, backup strategies, secure data handling

Prerequisites

1. **Basic Shell Commands**:
   - Familiarity with commands like ls, cd, cp, mv, rm, cat, and touch for basic file and directory operations.
2. **File Permissions and Ownership**:
   - Understanding of file permissions (chmod, chown) and the concepts of read, write, and execute access for users, groups, and others.
3. **Input and Output Redirection**:
   - Knowledge of input (<) and output (>, >>) redirection, and piping (|) for managing data flow in scripts.
4. **Text Processing Commands**:
   - Familiarity with text manipulation utilities like grep, awk, sed, and cut for searching and filtering file contents.
5. **Shell Scripting Basics**:
   - Understanding how to write, execute, and debug simple shell scripts, including using control structures like loops and conditionals.
6. **Error Handling**:
   - Basic knowledge of handling errors in scripts, such as checking if a file exists or if a command executed successfully.

Common Commands and Code Snippets:

Find Files and Directories

```
find /path/to/your/theme -name "*.txt"
```

Finds all .txt files in the specified directory.

Sort File Content

```
sort test1.txt > sorted_test1.txt
```

Sorts the contents of test1.txt and saves it to sorted_test1.txt.

Redirect Output to File

```
ls -l > directory_list.txt
```

Redirects the output of ls -l to directory_list.txt.

File Permission Change

```
chmod 600 test1.txt
```

Sets read and write permissions for the owner only.

Find and Remove Files Older Than 30 Days

```
find /path/to/your/theme -type f -mtime +30 -exec rm {} \;
```

Finds and removes files older than 30 days in the theme directory.

Theme Based Questions

1. Create a directory structure to represent different components of your theme (e.g., sections for a library, categories for a store).
2. Write a script to search for all files in your theme-related directory that contain a specific keyword and display their paths.
3. Perform a file permission audit on your theme's directory and modify permissions to ensure proper security.
4. Write a script to compare two theme-related files and highlight the differences.
5. Automate the process of archiving theme-related files older than 30 days into a compressed archive.
6. Develop a program to list all files in your theme's directory with their size and last modified date.
7. Write a script to identify duplicate files in your theme's directory and delete unnecessary duplicates.
8. Perform sorting of files based on their size or name and save the sorted list in a new file.
9. Implement a program to count and display the number of files and subdirectories in your theme's directory.
10. Write a script to generate a daily report summarizing all changes in your theme's directory (e.g., newly added, modified, or deleted files).

Ex.no. 6:

Orchestrating Processes: The Symphony of System Calls and Control

Objective: To understand process creation, parent-child relationships, and process management using system calls such as fork(), getpid(), getppid(), and wait() in Unix/Linux. Students will learn to create child processes, manage their execution, and explore process identifiers.

Job Roles : System Programmer, Kernel Developer, Embedded Systems Engineer

Skills : Process creation, system-level programming, kernel interactions

Prerequisites

- Knowledge of Unix/Linux processes.
- Understanding of basic system calls (fork(), exec(), wait(), etc.).
- Familiarity with compiling and running C programs in a Unix/Linux environment.

Common Commands and Code Snippets:

Fork a Process

```
pid_t pid = fork();
if (pid == 0) {
   printf("Child process\n");
} else {
   printf("Parent process\n");
}
```

Creates a child process using fork().

Get Process ID

```
printf("PID of this process: %d\n", getpid());
```

Prints the Process ID (PID) of the current process.

Exit from Process

```
exit(0);
```

Terminates the current process.

Wait for Child Process

```
wait(NULL);
```

Parent process waits for the child process to finish.

Get Parent Process ID

```
printf("Parent PID: %d\n", getppid());
```

Prints the Parent Process ID (PPID) of the current process.

Experiment Content

1. **Introduction**
   - **Process Basics**: Define a process and its components (text, data, stack).
   - **System Calls**: Explain how fork(), getpid(), getppid(), exec(), and wait() work.

2. **Key System Calls Used**
   - **fork()**: Creates a new process (child).
   - **getpid()**: Retrieves the process ID of the current process.
   - **getppid()**: Retrieves the parent process ID of the current process.
   - **wait()**: Makes the parent process wait for the child process to complete.

3. **Programming Tasks**
   - **Task 1**: Write a program to create a child process using fork() and display the PIDs of both parent and child.
   - **Task 2**: Modify the program to make the parent process wait for the child to finish using wait().
   - **Task 3**: Use the exec() family of functions in the child process to execute a new program.
   - **Task 4**: Create a program that demonstrates multiple levels of processes (parent → child → grandchild).

4. **Expected Program Structure**
   - Start with including necessary headers (#include <stdio.h>, #include <unistd.h>, #include <sys/types.h>, #include <sys/wait.h>).
   - Use fork() to create a child process.
   - Implement conditional statements to distinguish between parent and child processes.
   - Display process IDs using getpid() and getppid().
   - Use wait() to synchronize parent and child execution.
   - Optionally, demonstrate the use of exec() for executing a new program.

5. **Sample Code**

Basic Fork Example:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
   pid_t pid = fork();

   if (pid < 0) {
      perror("Fork failed");
      return 1;
   } else if (pid == 0) {
      // Child process
      printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());
```

```
    } else {
        // Parent process
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
        wait(NULL); // Wait for the child to finish
    }

    return 0;
}
```

Using Exec Example:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) {
        // Child process executes a new program
        printf("Child process: Executing 'ls' command\n");
        execlp("ls", "ls", NULL);
    } else {
        // Parent process
        printf("Parent process: Waiting for child to complete\n");
        wait(NULL);
        printf("Parent process: Child completed\n");
    }

    return 0;
}
```

6. **Hands-On Activities**
   - Create a process tree where the parent creates multiple children, and each child performs a specific task.
   - Measure the time it takes for the child process to execute using time.
   - Use getpriority() and setpriority() to modify the priority of a process.
7. **Troubleshooting Tips**
   - Ensure fork() returns successfully (check for negative values).
   - Use wait() correctly to avoid zombie processes.
   - Understand the limitations of exec() (replaces the current process image).

Theme Based Questions

1. Write a program to create a child process that performs a specific task related to your theme, like issuing a book or updating stock.

2. Create a multi-process program where each process handles a different operation in your theme (e.g., cataloging, borrowing, or returning).

3. Use fork() to create child processes and print the PID and PPID for each. Log these details in a theme-related file.
4. Implement a program to track and display the status of theme-related processes (e.g., running, stopped, or terminated).
5. Write a script to terminate a specific process related to your theme using its PID.
6. Develop a program to create a zombie process and explain its significance in your theme context.
7. Simulate parallel execution of theme-related tasks using multiple processes.
8. Write a program to calculate the total execution time of theme-related processes.
9. Create a script to generate a report of all running processes and identify those associated with your theme.
10. Implement inter-process communication between parent and child processes to pass data relevant to your theme.

Ex.no. 7:
Strategizing Execution: Process Scheduling for Peak System Efficiency

Objective: To understand and analyze the concept of process scheduling in Unix/Linux, including different scheduling policies, priority handling, and time sharing. Students will simulate scheduling scenarios and observe the behavior of processes under various scheduling algorithms.

Job Roles : Performance Analyst, OS Engineer, Systems Architect

Skills : Task scheduling, performance tuning, real-time system optimization

Prerequisites

- Basic understanding of processes and their states (new, ready, running, waiting, terminated).
- Familiarity with system calls (fork(), exec(), wait(), getpriority(), setpriority()).
- Knowledge of scheduling policies (First-Come-First-Serve (FCFS), Round-Robin (RR), Shortest Job First (SJF), etc.).
- Ability to compile and execute C programs in a Unix/Linux environment. scheduling algorithms.

Common Commands and Code Snippets:

View Running Processes

```
ps -aux
```

Lists all running processes with detailed information.

Kill a Process

```
kill -9 <PID>
```

Terminates a process by its PID.

Set Process Priority (Nice)

```
nice -n 10 my_process
```

Sets the priority of a process when starting it (lower values mean higher priority).

Display Process Tree

```
pstree
```

Displays a tree-like diagram of running processes.

Change Process Priority (Renice)

```
renice -n -5 -p <PID>
```

Changes the priority of a running process with the given PID.

Experiment Content

1. **Introduction**
   - **What is Process Scheduling?**: Explain scheduling as the method the operating system uses to allocate CPU time to processes.
   - **Types of Schedulers**: Long-term, short-term, and medium-term schedulers.

- **Common Scheduling Policies**:
    - First-Come-First-Serve (FCFS).
    - Shortest Job First (SJF).
    - Round-Robin (RR).
    - Priority Scheduling.

2. **Key Concepts to Explore**
    - **Scheduling Priorities**: Use getpriority() and setpriority() system calls to assign and retrieve process priorities.
    - **Simulating Scheduling Policies**: Implement basic scheduling algorithms in C to understand their behavior.
    - **Time Slicing**: Demonstrate time sharing using a Round-Robin approach.
3. **Programming Tasks**
    - **Task 1**: Write a program to create multiple processes using fork() and assign different priorities using setpriority().
    - **Task 2**: Implement a Round-Robin scheduler in C that uses a time slice for each process.
    - **Task 3**: Simulate the behavior of First-Come-First-Serve (FCFS) scheduling using an array of processes.
    - **Task 4**: Modify process priorities dynamically and observe the impact using getpriority() and setpriority().
4. **Expected Program Structure**
    - Include necessary headers (#include <stdio.h>, #include <unistd.h>, #include <sys/types.h>, #include <sys/wait.h>, #include <sys/time.h>, #include <sys/resource.h>).
    - Use fork() to create processes and assign them tasks.
    - Implement algorithms for process scheduling.

<u>**Sample Code**</u>

Priority Scheduling Example:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <sys/resource.h>

int main() {
   pid_t pid = fork();

   if (pid < 0) {
      perror("Fork failed");
      return 1;
   } else if (pid == 0) {
      // Child process
      printf("Child process: PID = %d, Priority = %d\n", getpid(), getpriority(PRIO_PROCESS,
getpid()));
      setpriority(PRIO_PROCESS, getpid(), 10); // Set a higher priority
```

```
        printf("Child process: New Priority = %d\n", getpriority(PRIO_PROCESS, getpid()));
    } else {
        // Parent process
        printf("Parent process: PID = %d, Priority = %d\n", getpid(), getpriority(PRIO_PROCESS,
getpid()));
        wait(NULL); // Wait for child to complete
        printf("Parent process: Child completed.\n");
    }

    return 0;
}
```

Round-Robin Simulation Example (Pseudo-code):

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>

#define NUM_PROCESSES 5
#define TIME_SLICE 2 // seconds

void simulate_process(int id) {
    printf("Process %d executing...\n", id);
    sleep(TIME_SLICE);
    printf("Process %d finished time slice.\n", id);
}

int main() {
    for (int i = 1; i <= NUM_PROCESSES; i++) {
        simulate_process(i);
    }
    return 0;
}
```

Theme Based Questions

1. Write a program to simulate Round-Robin scheduling for theme-related tasks, such as processing customer orders or library requests.
2. Demonstrate how priority scheduling can be used to prioritize critical tasks in your theme, like high-priority orders or overdue book returns.
3. Implement a program to compare the performance of different scheduling algorithms for your theme's operations.
4. Simulate task queues for your theme, showing how tasks are executed based on scheduling policies.
5. Create a program to measure and log the response times of tasks related to your theme.
6. Write a script to simulate load balancing between processes for theme-related operations.
7. Analyze the throughput of a scheduling algorithm when applied to tasks in your theme.
8. Create a program to implement multi-level queue scheduling for tasks related to your theme.

9. Write a script to display the execution timeline of scheduled processes for theme-related activities.
10. Demonstrate the impact of context switching on theme-related tasks by simulating multiple processes.

Ex.no. 8:

Bridging the Divide: Seamless Inter-Process Communication with Pipes

Objective: To understand and implement Inter-Process Communication (IPC) mechanisms using pipes. Students will learn how to create and use pipes for data transfer between related processes in a Unix/Linux environment.

Job Roles : Backend Developer, Middleware Engineer, Software Integration Engineer

Skills : Data communication, IPC techniques, system integration

Prerequisites

- Understanding of processes and their relationships (parent-child process model).
- Familiarity with system calls such as fork(), write(), read(), and pipe().
- Basic knowledge of file descriptors and their role in Unix/Linux systems.
- Proficiency in C programming and compiling C programs in a Unix/Linux environment.

Common Commands and Code Snippets:

Create a Pipe

```
int pipefd[2];
pipe(pipefd);
```

Creates a pipe for communication between processes.

Write to Pipe

```
write(pipefd[1], "Data", sizeof("Data"));
```

Writes data to the pipe.

Read from Pipe

```
char buffer[100];
read(pipefd[0], buffer, sizeof(buffer));
```

Reads data from the pipe.

```
**Parent and Child
Process with Pipe**
if (fork() == 0) {
close(pipefd[0]);
write(pipefd[1], "Message from child", 18);
} else {
close(pipefd[1]);
read(pipefd[0], buffer, sizeof(buffer));
printf("Received: %s\n", buffer);
}
```

Creates a parent-child relationship using a pipe for communication.

Using Pipe with External Commands

```
ls | grep "text"
```

Uses a pipe to list files and filter results using grep.

Experiment Content

1. **Introduction**
   - **What is IPC?**: Explain the need for communication between processes and how IPC facilitates it.
   - **Pipes**: Describe how pipes enable one-way communication between processes using file descriptors.
   - **Pipe Operations**: Explain the creation (pipe()), reading (read()), and writing (write()) mechanisms.
2. **Key Concepts**
   - **Types of Pipes**: Ordinary pipes (unidirectional, between related processes) and named pipes (FIFOs).
   - **File Descriptors**: Use of read and write ends of a pipe.
   - **Synchronization**: Managing the sequence of read and write operations.
3. **Programming Tasks**
   - **Task 1**: Create a pipe and demonstrate data transfer from a parent process to a child process.
   - **Task 2**: Implement a program where the child sends a message to the parent using the pipe.
   - **Task 3**: Extend the program to perform bidirectional communication using two pipes.
   - **Task 4**: Experiment with sending larger data chunks and observe how the pipe handles it.

Sample Code

Unidirectional Pipe Example:

```c
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int pipefd[2]; // Array to hold the file descriptors for the pipe
    pid_t pid;
    char message[] = "Hello from parent!";
    char buffer[100];

    // Create a pipe
    if (pipe(pipefd) == -1) {
        perror("Pipe failed");
        return 1;
    }

    pid = fork(); // Create a child process
    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) { // Child process
        close(pipefd[1]); // Close write end of the pipe
        read(pipefd[0], buffer, sizeof(buffer)); // Read from the pipe
        printf("Child received: %s\n", buffer);
        close(pipefd[0]); // Close read end of the pipe
    } else { // Parent process
        close(pipefd[0]); // Close read end of the pipe
        write(pipefd[1], message, strlen(message) + 1); // Write to the pipe
        close(pipefd[1]); // Close write end of the pipe
        wait(NULL); // Wait for child to complete
    }

    return 0;
}
```

Bidirectional Communication Example:

```
#include <stdio.h>
#include <unistd.h>
#include <string.h>

int main() {
    int pipe1[2], pipe2[2];
    pid_t pid;
    char parentMessage[] = "Hello from parent!";
    char childMessage[] = "Hello from child!";
    char buffer[100];

    // Create two pipes
    pipe(pipe1); // For parent to child
    pipe(pipe2); // For child to parent

    pid = fork();

    if (pid < 0) {
        perror("Fork failed");
        return 1;
    } else if (pid == 0) { // Child process
        close(pipe1[1]); // Close write end of pipe1
        close(pipe2[0]); // Close read end of pipe2

        read(pipe1[0], buffer, sizeof(buffer)); // Read from pipe1
        printf("Child received: %s\n", buffer);

        write(pipe2[1], childMessage, strlen(childMessage) + 1); // Write to pipe2

        close(pipe1[0]);
        close(pipe2[1]);
    } else { // Parent process
        close(pipe1[0]); // Close read end of pipe1
        close(pipe2[1]); // Close write end of pipe2

        write(pipe1[1], parentMessage, strlen(parentMessage) + 1); // Write to pipe1

        read(pipe2[0], buffer, sizeof(buffer)); // Read from pipe2
        printf("Parent received: %s\n", buffer);

        close(pipe1[1]);
        close(pipe2[0]);
        wait(NULL); // Wait for child to complete
    }

    return 0;
}
```

Theme Based Questions

1. Write a program where parent and child processes communicate theme-related data (e.g., book records or order details) using unnamed pipes.
2. Use named pipes (FIFOs) to exchange theme-related messages between two unrelated processes.
3. Implement a program to split a large theme-related file into smaller chunks and process them using pipes.
4. Write a script to simulate a producer-consumer model using pipes, where one process produces theme-related data and another consumes it.
5. Create a program to transmit a theme-related command from one process to another using pipes.
6. Write a script to encrypt and send data from one process to another using pipes.
7. Simulate a file transfer operation between processes using pipes for theme-related files.
8. Implement a program where one process sorts a list of theme-related items and passes the sorted data to another process.
9. Write a program to count the number of records in a theme-related file using communication between two processes via pipes.
10. Demonstrate bidirectional communication between two processes for theme-related activities using pipes.

Ex.no. 9:

Collaborative Processing: Synchronization with Semaphores and Shared Memory

Objective: To implement inter-process communication using semaphores and shared memory, enabling synchronized data exchange between processes. Students will learn how to manage shared resources and use semaphores for synchronization.

Job Roles : Distributed Systems Engineer, Parallel Processing Specialist, Software Engineer

Skills : Shared memory usage, process synchronization, multi-threading

Prerequisites

- Understanding of process synchronization and critical sections.
- Familiarity with IPC mechanisms, including semaphores and shared memory.
- Knowledge of system calls such as semget(), semctl(), shmget(), shmat(), shmdt(), and shmctl().

Content

1. **Introduction to Shared Memory**:
   - Create shared memory using shmget().
   - Attach to and detach from shared memory using shmat() and shmdt().
2. **Introduction to Semaphores**:
   - Use semaphores for synchronizing processes.
   - System calls for semaphore operations (semget(), semctl()).
3. **Hands-On Activities**:
   - Create two processes where the parent writes to shared memory, and the child reads it.
   - Use semaphores to ensure the proper sequence of operations.

Common Commands and Code Snippets:

Create Shared Memory Segment

```
key_t key = ftok("shmfile", 65);
int shmid = shmget(key, 1024, 0666|IPC_CREAT);
```

Creates a shared memory segment.

Attach Shared Memory

```
char *str = (char*) shmat(shmid, (void*)0, 0);
```

Attaches the shared memory segment to the process.

Semaphore Operations

```
sem_t *sem = sem_open("/sem_example", O_CREAT, 0666, 1);
sem_wait(sem); // Wait
sem_post(sem); // Signal
```

Controls access to shared resources using semaphores.

Detach Shared Memory

```
shmdt(str);
```

Detaches shared memory after use.

Destroy Shared Memory

```
shmctl(shmid, IPC_RMID, NULL);
```

Destroys the shared memory segment after use.

Theme Based Questions

1. Write a program to implement mutual exclusion for shared resources in your theme using semaphores.
2. Simulate a reader-writer problem related to your theme (e.g., accessing and updating book records) using semaphores and shared memory.
3. Create a shared memory segment where multiple processes write and read theme-related data. Synchronize access using semaphores.
4. Implement a program to calculate the total inventory or count of items in your theme using shared memory.
5. Demonstrate how semaphores can prevent deadlocks in theme-related operations.
6. Write a program to clear and delete a shared memory segment related to your theme.
7. Simulate concurrent updates to a shared resource in your theme, ensuring consistency using semaphores.
8. Develop a program to calculate and display aggregate data (e.g., total sales or borrowed books) using shared memory.
9. Write a script to implement and test semaphore operations (P and V) for theme-related tasks.
10. Create a program to log all read/write operations performed on shared memory related to your theme.

Ex.no. 10:

System Insight: Mastering Memory and Disk Resource Allocation

Objective: To explore memory and disk management in Unix/Linux systems. Students will analyze system memory usage, manage file system space, and understand disk quotas.

Job Roles : Storage Engineer, Virtualization Specialist, System Performance Engineer

Skills : Memory optimization, disk management, virtualization tools

Prerequisites

- Basic understanding of memory allocation and file systems.
- Familiarity with disk usage commands like df and du.

Content

1. **Memory Management**:
   - Use free and vmstat to monitor memory usage.
   - Analyze swap memory and buffers.
2. **Disk Management**:
   - Use df to analyze disk usage and available space.
   - Use du to analyze space used by files and directories.
   - Manage disk quotas with quota and edquota.
3. **Hands-On Activities**:
   - Monitor and report memory usage of processes.
   - Identify disk space usage and create reports.
   - Simulate disk quota restrictions for a user.

Common Commands and Code Snippets:

Check Disk Usage
```
df -h
```
Displays disk usage in human-readable format.

Check Memory Usage
```
free -h
```
Displays memory usage in human-readable format.

Clear Cache
```
sync; echo 3 > /proc/sys/vm/drop_caches
```
Clears page cache, dentries, and inodes.

View Disk Partitions
```
fdisk -l
```
Lists all disk partitions.

Mount a Disk

```
mount /dev/sdb1 /mnt
```

Mounts the disk /dev/sdb1 to the directory /mnt.

Theme Based Question

1. Write a script to display memory and disk usage statistics for files related to your theme.
2. Automate the cleanup of unnecessary files in your theme’s directory to free up disk space.
3. Create a program to simulate file fragmentation for your theme-related files and defragment them.
4. Write a script to monitor the memory consumption of theme-related processes and alert if it exceeds a threshold.
5. Simulate a memory paging system for theme-related operations and analyze the performance.
6. Develop a script to calculate the space occupied by each category or section in your theme.
7. Write a program to simulate virtual memory allocation for theme-related tasks.
8. Create a script to generate disk usage reports for theme-related directories and files.
9. Implement a program to monitor swap usage for theme-related processes.
10. Write a script to back up theme-related data and verify the integrity of the backup.

Ex.no 11:

Architecting Networks: Precision Configuration of IP Addressing

Objective: To configure network settings, including IP addressing, subnet masks, and gateways. Students will understand network configuration basics and test network connectivity.

Job Roles : Network Administrator, Network Engineer, Infrastructure Specialist

Skills : IP configuration, routing, network troubleshooting

Prerequisites

- Basic networking knowledge (IP addressing, subnetting).
- Familiarity with network commands like ifconfig, ip, ping, and netstat.

Content

1. **Static IP Configuration**:
   - Configure a static IP using ifconfig or ip addr add.
   - Set subnet masks and gateways.
2. **Dynamic IP Configuration**:
   - Use dhclient for DHCP configuration.
3. **Testing Connectivity**:
   - Use ping and traceroute to test network connectivity.
   - Use netstat or ss to monitor network connections.
4. **Hands-On Activities**:
   - Assign a static IP and test connectivity.
   - Configure and verify DNS settings.
   - Set up a network interface to communicate with another system.

Common Commands and Code Snippets:

View Current IP Address

```
ifconfig
```

Displays network interface configuration, including IP addresses.

Set Static IP Address

```
sudo ifconfig eth0 192.168.1.100 netmask 255.255.255.0 up
```

Sets a static IP address (192.168.1.100) for the network interface eth0.

View Network Routing Table

```
netstat -r
```

Displays the system's routing table.

Configure Network Interface Using nmcli

```
nmcli con mod eth0 ipv4.addresses 192.168.1.100/24
nmcli con mod eth0 ipv4.gateway 192.168.1.1
nmcli con up eth0
```

Uses nmcli to modify the IP address and gateway settings for eth0.

Display DNS Configuration

```
cat /etc/resolv.conf
```

Displays the system's DNS configuration.

Ping a Remote Host

```
ping -c 4 8.8.8.8
```

Pings Google's DNS server (8.8.8.8) to test connectivity.

Configure Default Gateway

```
sudo route add default gw 192.168.1.1
```

Sets the default gateway to 192.168.1.1.

Check Network Interface Status

```
ip link show
```

Displays the status of all network interfaces.

Assign Dynamic IP Using DHCP

```
sudo dhclient eth0
```

Requests an IP address from the DHCP server for the eth0 interface.

Display IP Configuration Using ip Command

```
ip addr show
```

Displays the IP address and other details for all network interfaces.

Theme Based Questions

1. Write a script to configure a static IP address for a machine related to your theme and verify the configuration.
2. Develop a program to simulate a client-server communication model related to your theme using assigned IP addresses.
3. Create a script to display and log the network configuration details of all machines related to your theme.
4. Write a script to automate the assignment of IP addresses to multiple machines related to your theme.
5. Test the connectivity between machines in your theme's network using tools like ping or traceroute.
6. Simulate a DNS resolution for machines related to your theme and log the results.
7. Configure a machine to act as a gateway for theme-related operations and test its functionality.
8. Write a script to troubleshoot and resolve common network configuration issues related to your theme.
9. Create a program to simulate dynamic IP allocation for theme-related machines using DHCP.
10. Write a script to log all network activity related to your theme.

Ex.no 12:

Fortifying Infrastructure: Advanced Firewall Configuration for Security

Objective: To configure firewall rules using tools like iptables, firewalld, or ufw. Students will learn how to secure a system by controlling incoming and outgoing traffic.

Job Roles: Cybersecurity Specialist, Network Security Engineer, IT Security Analyst

Skills: Firewall setup, access control, system hardening

Prerequisites

- Basic knowledge of firewalls and network security.
- Familiarity with ports, protocols, and access control lists (ACLs).

Content

1. **Firewall Basics**:
   - Understand the role of firewalls in network security.
   - Learn about input, output, and forwarding chains in iptables.
2. **Configuring Firewall Rules**:
   - Use iptables to allow or block specific ports.
   - Configure firewalld to manage zones and services.
3. **Testing Firewall Configuration**:
   - Verify rules using iptables -L or firewalld commands.
   - Use nc (netcat) to test port access.
4. **Hands-On Activities**:
   - Block and allow specific IP addresses and ports.
   - Set up rules for a basic web server (e.g., allow port 80).
   - Test firewall configurations for misconfigurations or loopholes.

Common Commands and Code Snippets:

Check Current Firewall Status

```
sudo ufw status
```

Displays the status of the Uncomplicated Firewall (UFW).

Enable UFW

```
sudo ufw enable
```

Enables the UFW to start filtering traffic.

Allow Incoming SSH Connections

```
sudo ufw allow ssh
```

Allows incoming SSH connections through the firewall.

Deny Incoming HTTP Connections

```
sudo ufw deny http
```

Denies incoming HTTP traffic (port 80).

Allow Specific Port (e.g., Port 8080)

```
sudo ufw allow 8080/tcp
```

Allows incoming traffic on port 8080.

Enable Firewall Logging

```
sudo ufw logging on
```

Enables logging of firewall events.

View UFW Firewall Rules

```
sudo ufw status verbose
```

Displays all active firewall rules with detailed information.

Disable UFW

```
sudo ufw disable
```

Disables the UFW firewall.

Allow IP Range

```
sudo ufw allow from 192.168.1.0/24 to any port 22
```

Allows SSH access from the 192.168.1.0/24 subnet.

Reset Firewall Configuration

```
sudo ufw reset
```

Resets the firewall configuration to its default state.

Theme Based Questions

1. Write a script to configure firewall rules to allow theme-related services and block others.
2. Demonstrate the use of iptables or firewalld to set up access controls for theme-related operations.
3. Create a script to log all access attempts to a theme-related service and analyze the logs.
4. Write a program to automate the configuration of firewall rules for theme-related servers.
5. Simulate an attack on a theme-related service and demonstrate how your firewall rules block unauthorized access.
6. Develop a script to monitor and log all traffic through theme-related ports.
7. Write a script to configure firewall rules to allow only specific IP addresses to access theme-related services.
8. Create a program to reset and restore firewall rules for theme-related configurations.
9. Write a script to simulate traffic filtering for theme-related network activities.
10. Test and validate your firewall configuration by attempting both allowed and restricted theme-related actions.