Practice exercises for students organized by experiment

Introduction to Linux Commands

Exercise 1

1. List all the files and directories in your home directory.
2. Create a new directory called "test" in your home directory.
3. Change into the "test" directory you just created.
4. Create a new file called "example.txt" in the "test" directory.
5. Open the "example.txt" file and write "Hello, World!" in it.
6. Save and exit the file.
7. List all the files in the "test" directory.
8. Rename the "example.txt" file to "sample.txt".
9. List all the files in the "test" directory again to verify the file has been renamed.
10. Remove the "test" directory and all its contents.
11. List all the files and directories in the root directory.
12. Change into the "/etc" directory.
13. List all the files and directories in the "/etc" directory.

Exercise 2

1. Create a new directory called "lab_exercises" in your home directory.
2. Inside the "lab_exercises" directory, create a new file called "file1.txt" and write some text in it.
3. Create a copy of "file1.txt" and name it "file2.txt" using the cp command.
4. Verify that "file2.txt" is an exact copy of "file1.txt" by opening both files and comparing their contents.
5. Create a new directory called "backup" inside the "lab_exercises" directory.
6. Move "file1.txt" and "file2.txt" to the "backup" directory using the mv command.
7. Verify that both files have been moved to the "backup" directory by listing its contents.
8. Create a new file called "file3.txt" in the "lab_exercises" directory and write some text in it.
9. Create a new directory called "archive" inside the "lab_exercises" directory.
10. Move "file3.txt" to the "archive" directory and rename it to "file3_backup.txt" using the mv command.
11. Verify that "file3_backup.txt" has been moved to the "archive" directory and that its contents are the same as "file3.txt".
12. Create a new directory called "temp" inside the "lab_exercises" directory.
13. Create a new file called "file4.txt" in the "temp" directory and write some text in it.
14. Move "file4.txt" to the "lab_exercises" directory using the mv command.
15. Verify that "file4.txt" has been moved to the "lab_exercises" directory and that its contents are the same as before.

Shell Programming

Exercise 1

A. Write a script that displays "Hello, World!" when executed.

B. Modify the script to accept a command line argument and display "Hello, <argument>!" instead of "Hello, World!".

Exercise 2

Write a script that accepts two command line arguments and displays their sum.

Exercise 3

A. Write a script that accepts a directory name as a command line argument and displays the number of files in that directory.

B. Modify the script to display the number of files in the directory and all its subdirectories.

Exercise 4

A. Write a script that accepts a filename as a command line argument and displays the number of lines, words, and characters in that file.

B. Modify the script to accept multiple file names as command line arguments and display the number of lines, words, and characters in each file.

Exercise 5

A. Declare a variable called "name" and assign your name to it. Display the value of the variable using the echo command.

B. Declare a variable called "age" and assign your age to it. Display the value of the variable using the echo command.

C. Declare a variable called "color" and assign your favorite color to it. Display the value of the variable using the echo command.

Exercise 6

Declare a variable called "num1" and assign the value 10 to it. Declare a second variable called "num2" and assign the value 5 to it. Add the values of the two variables and display the result using the echo command.

Exercise 7

A. Declare a variable called "filename" and assign the value "sample.txt" to it. Use the variable to create a new file with that name using the touch command.

B. Declare a variable called "directory" and assign the value "myfolder" to it. Use the variable to create a new directory with that name using the mkdir command.

Exercise 8

Declare a variable called "files" and assign a list of filenames to it. Use a loop to display the contents of each file in the list using the cat command.

Exercise 9

A. Write a command that displays the contents of a file called "file1.txt" on the screen.

B. Use input redirection to create a new file called "file2.txt" with the contents of "file1.txt".

C. Write a command that appends the contents of "file1.txt" to the end of "file2.txt".

Exercise 10

A. Write a for loop that prints the numbers from 1 to 10 on the screen.

B. Modify the loop to print only the even numbers from 1 to 10.

Exercise 11

A. Write a loop that displays the names of all files in the current directory.

B. Modify the loop to display only the names of files with the extension ".txt".

Exercise 12

A. Write a case/esac statement that displays a message on the screen based on the value of a variable called "day". If the value is "Monday", the message should be "It's the start of the week". If the value is "Friday", the message should be "Thank goodness it's Friday!". For any other value, the message should be "Just another day".

B. Modify the case/esac statement to use a read command to read the value of "day" from the user.

Exercise 13

Write a case/esac statement that calculates the area of a geometric shape based on the user's input. If the input is "square", the statement should ask the user for the length of the side and display the area. If the input is "rectangle", the statement should ask the user for the length and width and display the area. If the input is "circle", the statement should ask the user

Exercise 14

A. Write an if statement that checks if a variable called "x" is greater than 10. If it is, display the message "x is greater than 10".

B. Modify the if statement to check if "x" is equal to 10 as well. If it is, display the message "x is equal to 10".

File Manipulation using System Calls

Exercise 1

Write a program in C that creates a file called "output.txt", writes the text "Hello, World!" to it, and then closes the file.

Exercise 2

Write a program in C that reads the contents of a file called "input.txt" and writes them to a new file called "output.txt". You should use system calls like open(), read(), and write().

Exercise 3

Write a program in C that reads a file called "input.txt" and counts the number of lines in the file. You should use system calls like open(), read(), and write().

Exercise 4

A. Write a C program that creates a file called "numbers.txt" and writes 100 integers to it, one integer per line.

B. Using the lseek system call, move the file pointer to the beginning of the file.

C. Read the first 10 integers from the file and print them to the console.

Exercise 5

Write a C program that prints the last 10 characters of a file named as "input.txt" on the screen. Use open, read, write and lseek system calls.

Exercise 6

Write a C program that prints half content of a file named as "input.txt" on the screen. Use open, read, write and lseek system calls. If there are 100 characters written in the file, your program should display the first 50 characters on the screen.

Directory Manipulation using System Calls

Exercise 1

A. Write a C program that opens a directory called "my_directory" and reads all the files and directories inside it.

B. For each file and directory, print its name and whether it is a file or a directory.

C. Count the number of files and directories inside the "my_directory" directory.

D. Close the directory.

Process Management using System Calls

Exercise 1
Write a C program that uses the fork system call to create a child process. In the child process, print the process ID.(PID) and the parent process ID (PPID). In the parent process, print the PID and the child's PID.

Exercise 2
Write a C program that uses the fork system call to create a child process. In the child process, print a message indicating that it is the child process. In the parent process, print a message indicating that it is the parent process.

Exercise 3
Write a C program that uses the fork system call to create a child process. In the child process, create a file called "child.txt" and write the message "This is the child process" to it. In the parent process, create a file called "parent.txt" and write the message "This is the parent process" to it.

Exercise 4
Write a C program that uses the fork system call to create a child process. In the child process, print the sum of the first 100 positive integers. In the parent process, print the sum of the first 1000 positive integers.

Creation of Multithreaded Processes using Pthread Library

Exercise 1
Write a C program that creates two threads using the Pthread library. Each thread should print its thread ID to the console.

Exercise 2
Write a C program that creates two threads using the Pthread library. One thread should generate a random number, print it to the console and another should check for a prime number.

Exercise 3
Write a C program that creates two threads using the Pthread library. One thread should print even numbers from 2 to 100, and the other thread should print odd numbers from 1 to 99.

Process Synchronization using Semaphores/Mutex

Exercise 1
Write a C program to create two threads that increment a shared variable using a mutex to synchronize access to the variable.

Exercise 2
Write a C program to create two processes that increment a shared variable using semaphores to synchronize access to the variable.

Exercise 3

Write a C program to create two processes that implement a producer-consumer model using semaphores to synchronize access to the shared buffer.

Inter Process Communication using Pipes/Shared Memory/RPC

Exercises on pipe(), dup()/dup2()

Exercise 1
Write a C program that creates a file called "file1.txt" and writes some text to it. Then, use the dup system call to create a duplicate file descriptor for the file. Finally, use the duplicate file descriptor to write some more text to the file.

Exercise 2
Write a C program that creates two pipes using the pipe system call. Then, fork a child process. In the parent process, use the dup2 system call to redirect the standard input to one end of the pipe, and the standard output to the other end of the pipe. In the child process, read from the standard input and write to the standard output.

Exercise 3
Write a C program that takes a file name as a command-line argument. Use the open, dup, and dup2 system calls to open the file, create two duplicate file descriptors for it, and redirect the standard input and standard output to those file descriptors. Then, read from the standard input and write to the standard output.

Exercise 4
Write a C program that creates a file called "file1.txt" and writes some text to it. Then, use the dup2 system call to create a duplicate file descriptor for the file, and close the original file descriptor. Finally, use the duplicate file descriptor to read the text from the file and write it to the standard output.

Exercises on mkfifo(): Named pipe

Exercise 1
Write a C program that creates a named pipe (FIFO) using the mkfifo system call. Then, fork a child process. In the parent process, write some data to the pipe using the write system call. In the child process, read the data from the pipe using the read system call.

Exercise 2
Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, use the open system call to open the pipe for writing. Write some data to the pipe using the write system call. Finally, use the cat command to read the data from the pipe.

Exercise 3
Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, use the open system call to open the pipe for reading. Read some data from the pipe using the read system call. Finally, use the echo command to write the data to the standard output.

Exercise 4

Write a C program that creates a named pipe called "mypipe" using the mkfifo system call. Then, fork a child process. In the parent process, use the open system call to open the pipe for writing. In the child process, use the open system call to open the pipe for reading. Then, write some data to the pipe in the parent process and read the data from the pipe in the child process.

Exercise 5

Write a program in C that creates a child process using fork(). The parent process should read a message from the user and send it to the child process using a pipe. The child process should then read the message from the pipe and print it to the console.

Shared Memory

Exercise 1

Implement a simple message passing system using shared memory in C. Create a parent process that forks a child process and communicates with it using the shared memory. Use the shmget, shmat, and shmdt functions to create a shared memory segment, attach to it, and detach from it. The parent process should write a message to the shared memory segment, and the child process should read and print the message.