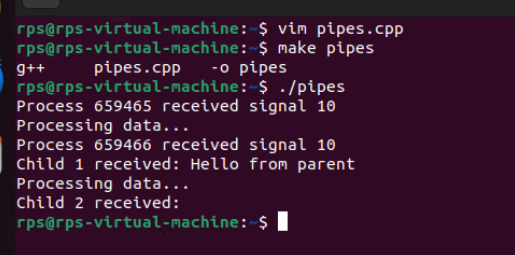
**Problem Statement: Signal Handling and Inter-Process Communication using Pipes in C++**

**Design and implement a robust system in C++ that effectively utilizes signals to control the behavior of multiple processes and employs pipes for inter-process communication, enabling coordinated data exchange and process synchronization.**



#include <iostream>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <signal.h>

#include <cstring>

using namespace std;

const int BUFFER\_SIZE = 100;

void signalHandler(int signum) {

cout << "Process " << getpid() << " received signal " << signum << endl;

if (signum == SIGUSR1) {

cout << "Processing data..." << endl;

}

}

int main() {

int pipefd[2];

if (pipe(pipefd) == -1) {

perror("pipe");

return 1;

}

signal(SIGUSR1, signalHandler); // Set up signal handler

pid\_t pid1 = fork();

if (pid1 == 0) { // Child 1

close(pipefd[1]); // Close write end

char buffer[BUFFER\_SIZE];

read(pipefd[0], buffer, BUFFER\_SIZE);

cout << "Child 1 received: " << buffer << endl;

close(pipefd[0]);

return 0;

}

pid\_t pid2 = fork();

if (pid2 == 0) { // Child 2

close(pipefd[1]); // Close write end

char buffer[BUFFER\_SIZE];

read(pipefd[0], buffer, BUFFER\_SIZE);

cout << "Child 2 received: " << buffer << endl;

close(pipefd[0]);

return 0;

}

// Parent process

close(pipefd[0]); // Close read end

const char \*message = "Hello from parent";

write(pipefd[1], message, strlen(message) + 1); // Send message to children

close(pipefd[1]); // Close write end after sending

// Send signals to children to process the data

kill(pid1, SIGUSR1);

kill(pid2, SIGUSR1);

// Wait for children to finish

waitpid(pid1, nullptr, 0);

waitpid(pid2, nullptr, 0);

return 0;

}

**Problem Statement: File Management Script with Functions and Arguments**

**Objective**

**Create a shell script that manages files in a specified directory. The script should include functions to perform the following tasks:**

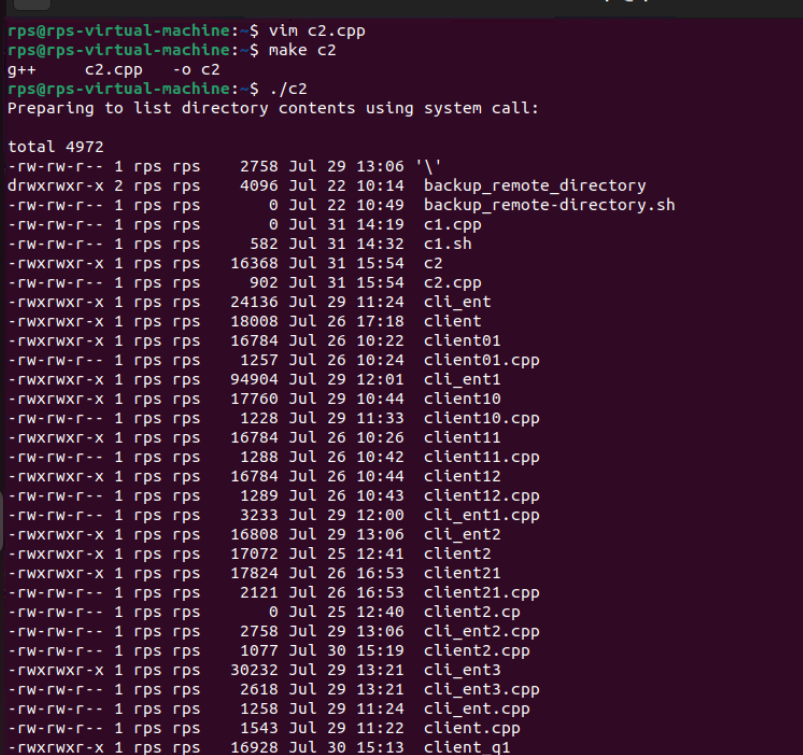
**List all files in the directory.**

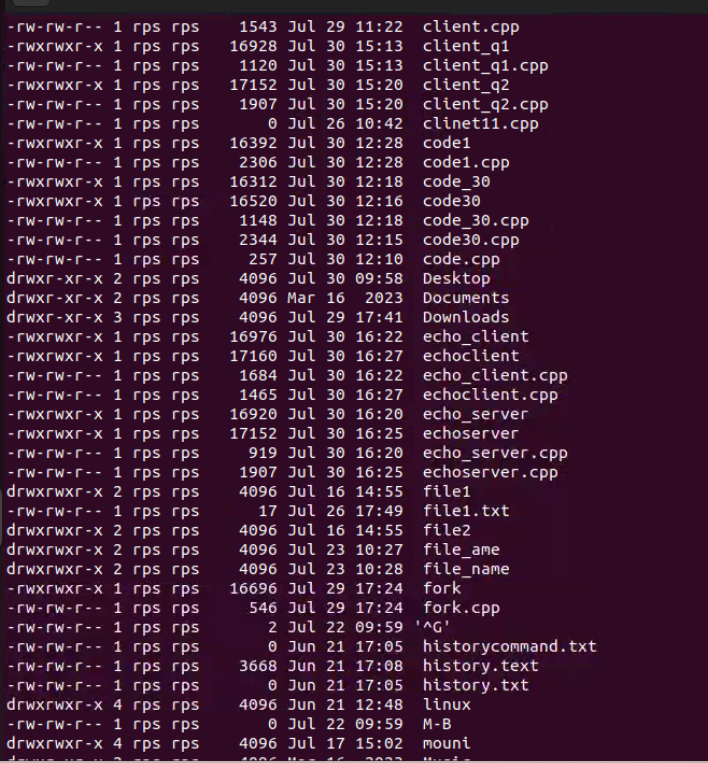
**Display the total number of files.**

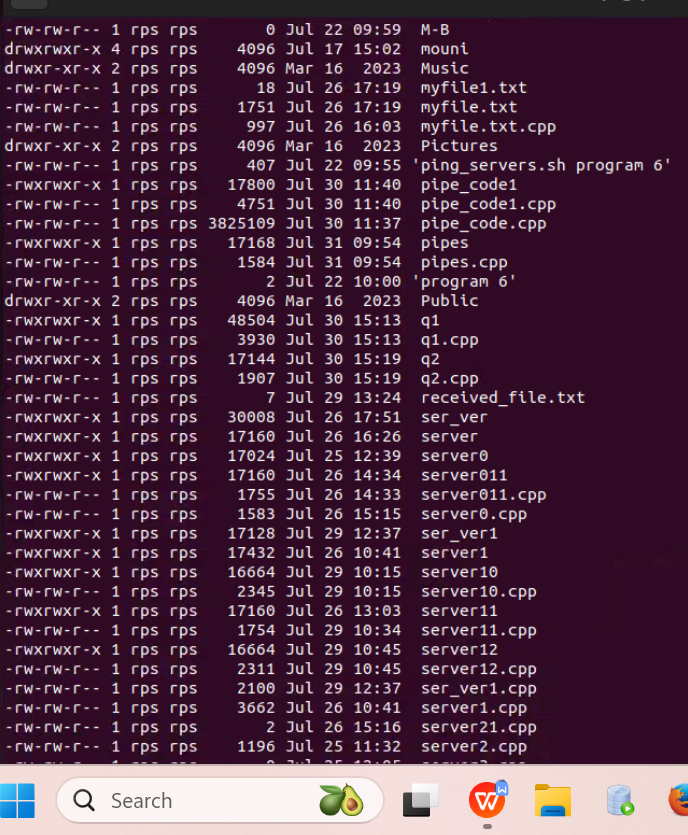
**Copy a specified file to a new location.**

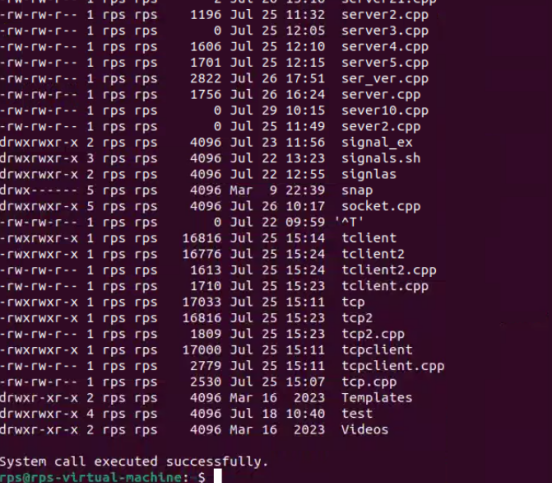
**Move a specified file to a new location.**

**Delete a specified file.**

****

****

****

****

#include <iostream> // For std::cout

#include <cstdlib> // For system() function

#include <unistd.h> // For write() system call

#include <cstring> // For strlen()

int main() {

// Define messages

const char\* preMessage = "Preparing to list directory contents using system call:\n\n";

const char\* postMessage = "\nSystem call executed successfully.\n";

const char\* errorMessage = "System call failed.\n";

// Print the preMessage using write()

write(STDOUT\_FILENO, preMessage, strlen(preMessage));

// Invoke the system call to list directory contents

int result = system("ls -l");

// Check the result of the system call

if (result == -1) {

write(STDERR\_FILENO, errorMessage, strlen(errorMessage));

return 1;

}

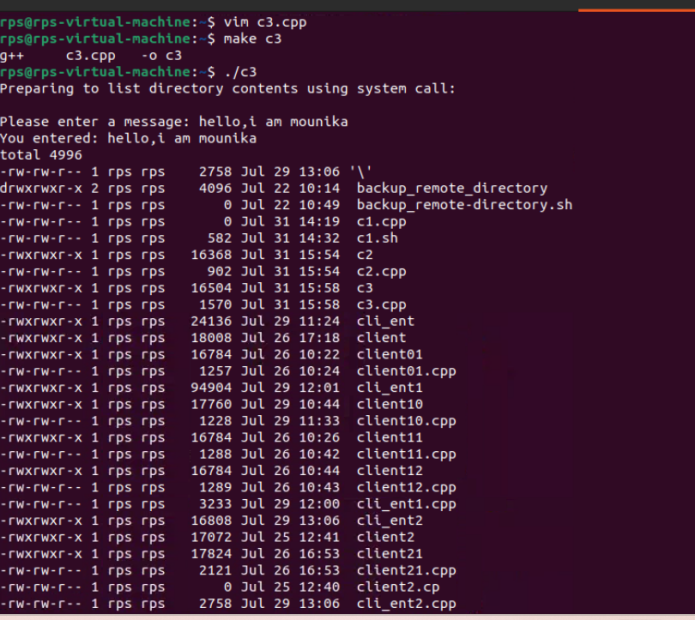
// Print the postMessage using write()

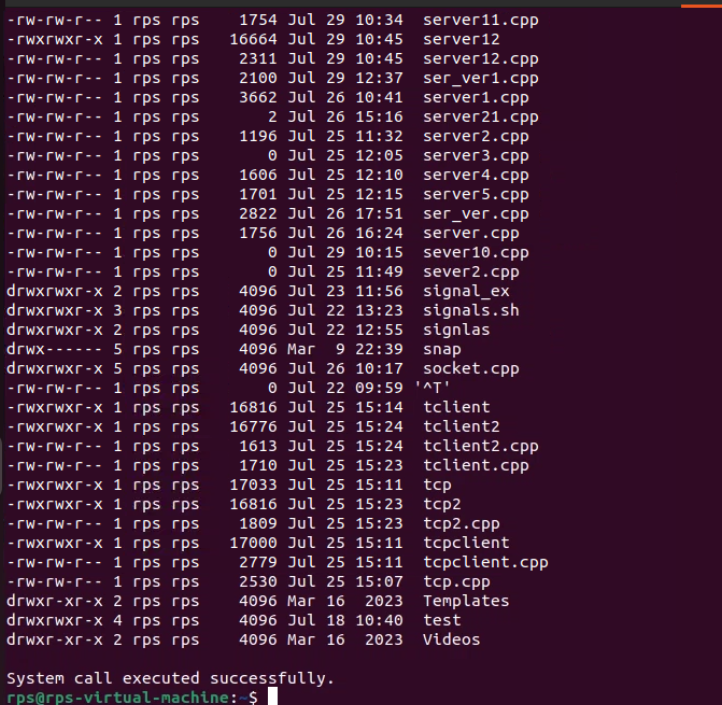
write(STDOUT\_FILENO, postMessage, strlen(postMessage));

return 0;

}

**after writing the code please read from user and write on screen using read and write apis in cpp**

****

****

#include <iostream>

#include <unistd.h> // For read() and write() system calls

#include <cstring> // For strlen()

#include <cstdlib> // For system() function

int main() {

// Define buffer size and create buffer

const int bufferSize = 256;

char buffer[bufferSize];

// Define messages

const char\* preMessage = "Preparing to list directory contents using system call:\n\n";

const char\* postMessage = "\nSystem call executed successfully.\n";

const char\* errorMessage = "System call failed.\n";

const char\* userPrompt = "Please enter a message: ";

// Print the preMessage using write()

write(STDOUT\_FILENO, preMessage, strlen(preMessage));

// Prompt the user to enter a message

write(STDOUT\_FILENO, userPrompt, strlen(userPrompt));

// Read the user input

ssize\_t bytesRead = read(STDIN\_FILENO, buffer, bufferSize - 1);

if (bytesRead < 0) {

perror("Read error");

return 1;

}

// Null-terminate the buffer to make it a proper string

buffer[bytesRead] = '\0';

// Print the user input back to the screen

write(STDOUT\_FILENO, "You entered: ", 13);

write(STDOUT\_FILENO, buffer, strlen(buffer));

// Invoke the system call to list directory contents

int result = system("ls -l");

// Check the result of the system call

if (result == -1) {

write(STDERR\_FILENO, errorMessage, strlen(errorMessage));

return 1;

}

// Print the postMessage using write()

write(STDOUT\_FILENO, postMessage, strlen(postMessage));

return 0;

}

**Problem Statement: File Operations using System Calls in C++**

**Description:**

**Write a C++ program that performs various file operations using Linux system calls. The program should create a file, write to it, read from it, and then delete the file. The program should handle errors appropriately and ensure proper resource management (e.g., closing file descriptors).**

**Instructions:**

**Create a File:**

**Use the open system call to create a new file named "example.txt" with read and write permissions.**

**If the file already exists, truncate its contents.**

**Write to the File:**

**Write the string "Hello, World!" to the file using the write system call.**

**Ensure that all bytes are written to the file.**

**Read from the File:**

**Use the lseek system call to reset the file pointer to the beginning of the file.**

**Read the contents of the file using the read system call and store it in a buffer.**

**Print the contents of the buffer to the standard output.**

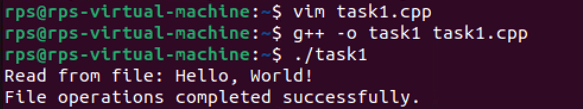
**Delete the File:**

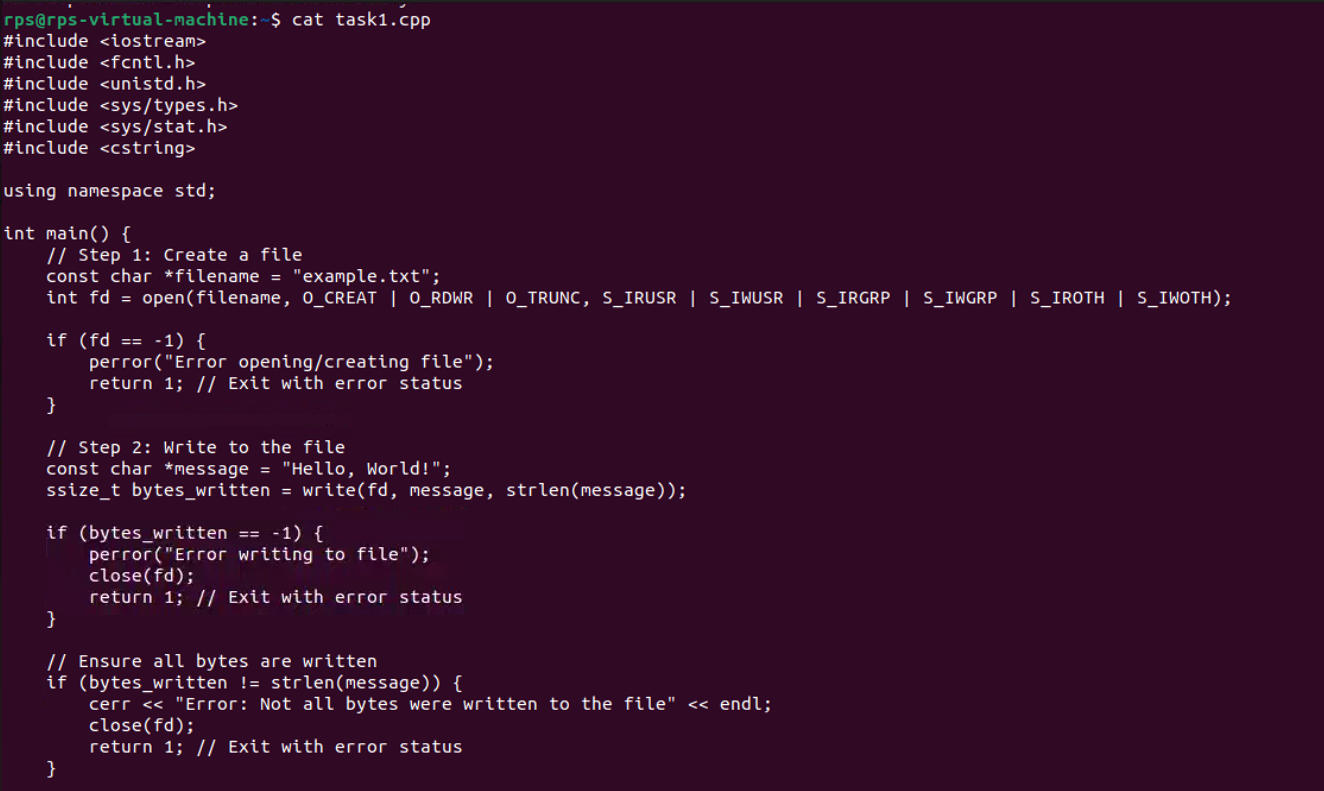
**Close the file descriptor using the close system call.**

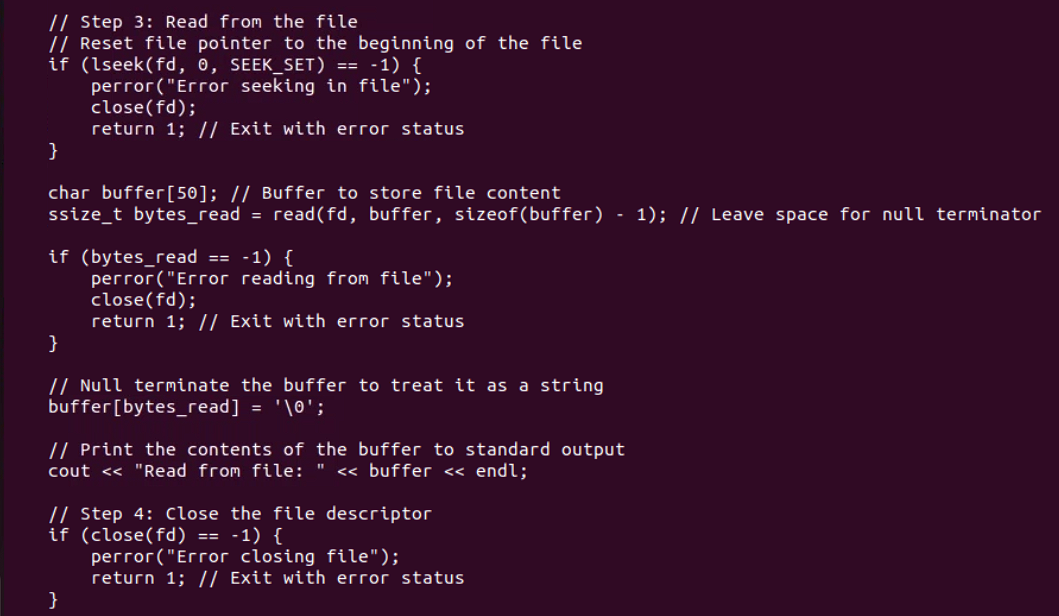
**Use the unlink system call to delete the file "example.txt".**

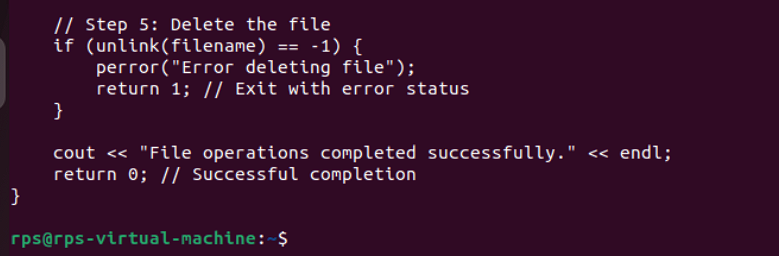
**Error Handling:**

**Ensure proper error handling for each system call. If a system call fails, print an error message and exit the program with a non-zero status.**

****

****

****

****

b#!/bin/bash

echo "simple calculator"

sum=0

i="y"

echo "enter first number"

read n1

echo "enter second number"

read n2

while [ $i = "y" ]

do

echo "1.Addition"

echo "2.Subtraction"

echo "3.Multiplication"

echo "4.Division"

echo "Enter choice"

read ch

case $ch in

1)sum=$(echo " $n1 + $n2" | bc -l)

echo "Addition is =" $sum;;

2)sum=$(echo "$n1 - $n2" | bc -l)

echo "Sub is =" $sum;;

3)sum=$(echo "$n1 \* $n2" | bc -l)

echo "Mul is =" $sum;;

4)sum=$(echo "$n1 / $n2" | bc -l)

echo "div is =" $sum;;

\*)echo "invalid choice"

esac

echo "Do you want to continue"

read i

if [ $i != "y" ]

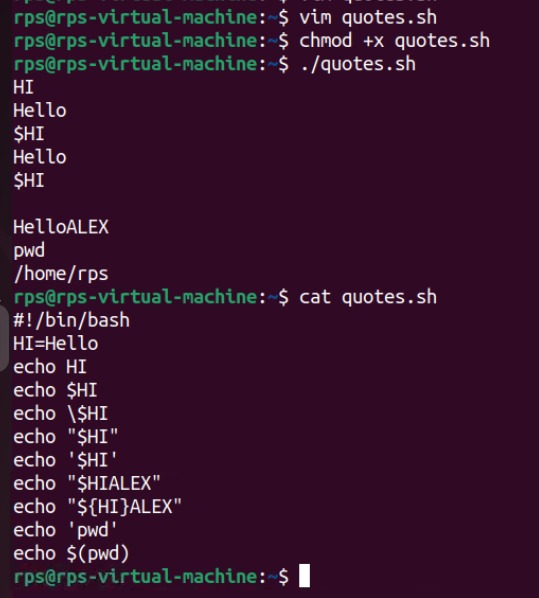
then

exit

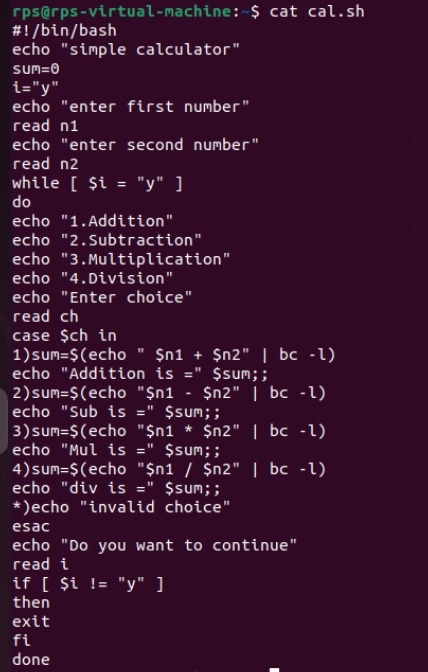
fi

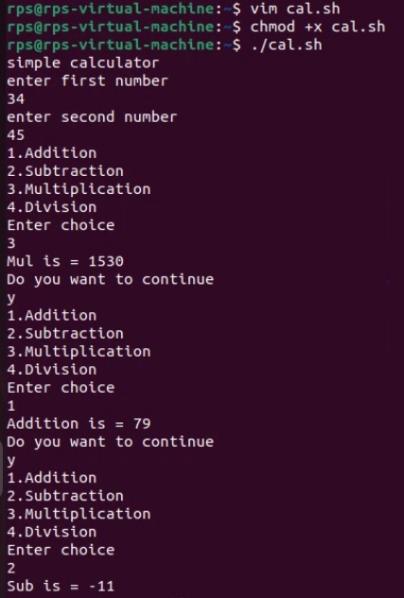
done

**Shell Programs:**



**Calculator:**





**Change File Permissions**

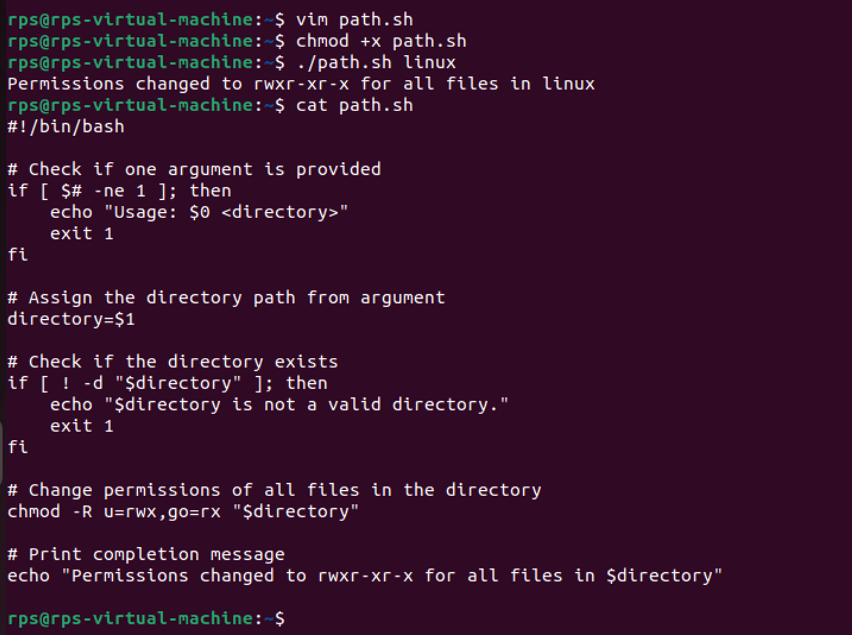
**Description: Write a shell script that takes a directory path as an argument and changes the permissions of all files within that directory to read, write, and execute for the owner, and read and execute for the group and others.**

**Instructions:**

**The script should accept one argument, the directory path.**

**Change permissions of all files in the specified directory to rwxr-xr-x.**

**Print a message indicating the completion of the permission change.**

****

**Problem 2: Count Files and Directories**

**Description: Write a shell script that counts the number of files and directories in a given directory.**

**Instructions:**

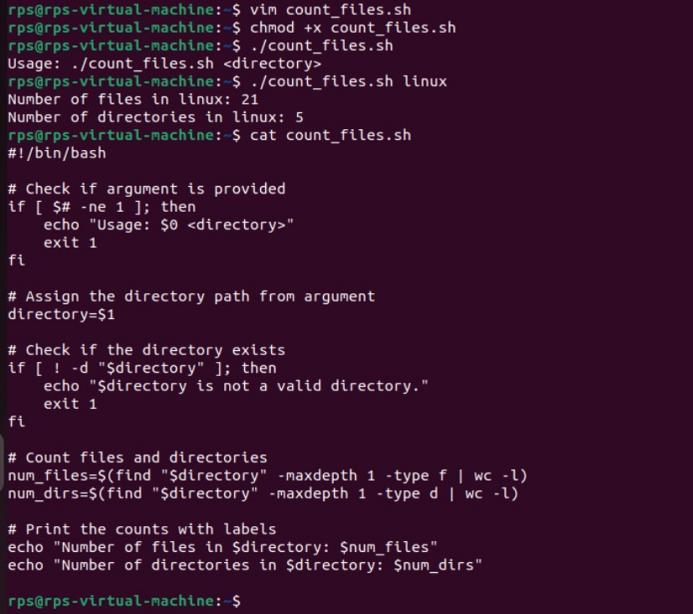
**The script should accept one argument, the directory path.**

**Count the number of files and directories separately.**

**Print the counts with appropriate labels.**

**Sample Input:**

**./count\_files\_dirs.sh /path/to/directory**



**Problem 3: Find and Replace Text in Files**

**Description: Write a shell script to search for a specific text string in all files within a directory and replace it with another string.**

**Instructions:**

**The script should accept three arguments: directory path, search string, and replacement string.**

**Search for the specified string in all files within the directory.**

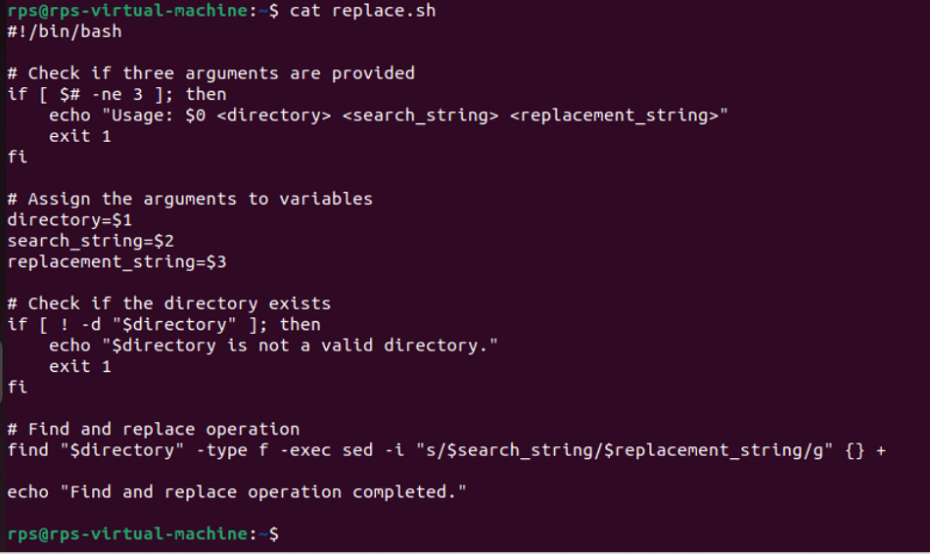
**Replace the string with the given replacement string in all occurrences.**

**Print a message indicating the completion of the find and replace operation.**

**Sample Input:**

**./find\_replace.sh /path/to/directory "old\_text" "new\_text"**

****

****

**Problem 4: Disk Usage Report**

**Description: Write a shell script that generates a report of disk usage for a specified directory.**

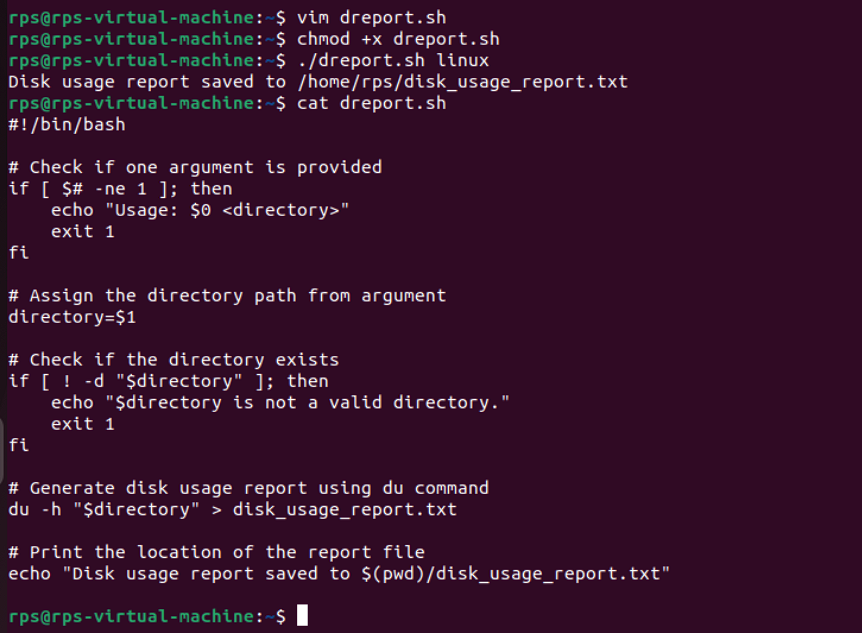
**Instructions:**

**The script should accept one argument, the directory path.**

**Use the du command to generate a disk usage report for the directory.**

**Save the report to a file named disk\_usage\_report.txt in the current directory.**

**Print a message indicating where the report is saved.**

****