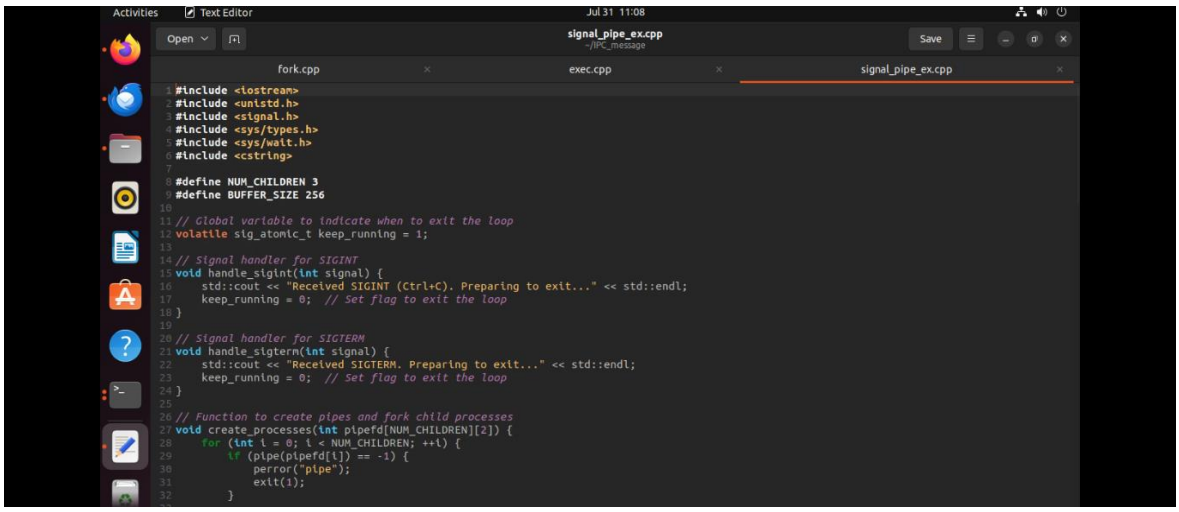
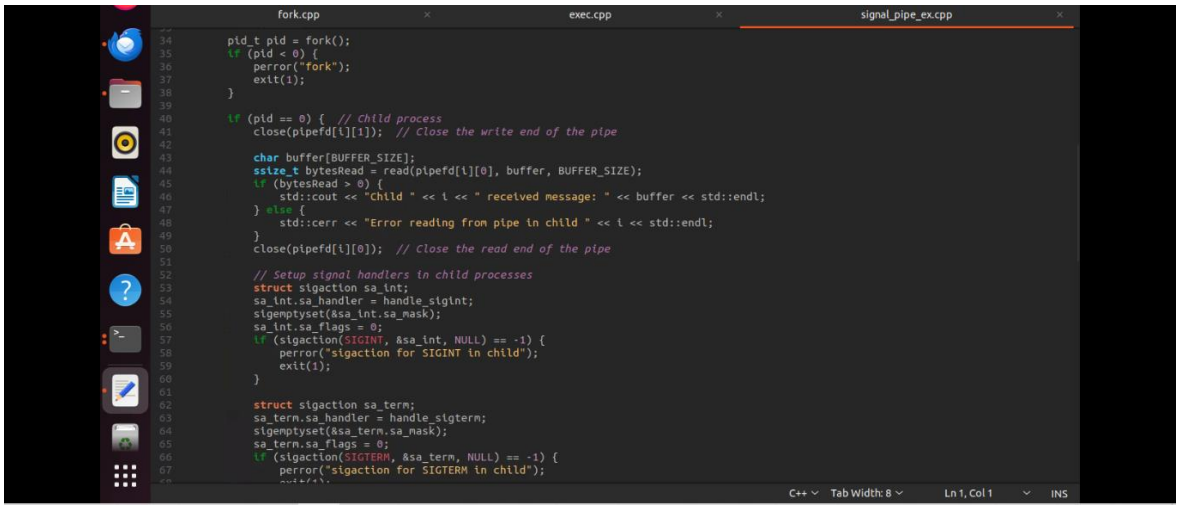
Day -12 LSP Assignment (Task – 1)

1. **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.





```cpp
            // Wait for signals
            while (keep_running) {
                sleep(1);
            }

            std::cout << "Child " << i << " exiting..." << std::endl;
            _exit(0);
        } else {  // Parent process
            close(pipefd[i][0]);  // Close the read end of the pipe
        }
    }
}

int main() {
    int pipefd[NUM_CHILDREN][2];  // Array of pipe file descriptors

    // Setup signal handlers in the parent process
    struct sigaction sa_int;
    sa_int.sa_handler = handle_sigint;
    sigemptyset(&sa_int.sa_mask);
    sa_int.sa_flags = 0;
    if (sigaction(SIGINT, &sa_int, NULL) == -1) {
        perror("sigaction for SIGINT in parent");
        return 1;
    }

    struct sigaction sa_term;
    sa_term.sa_handler = handle_sigterm;
    sigemptyset(&sa_term.sa_mask);
    sa_term.sa_flags = 0;
    if (sigaction(SIGTERM, &sa_term, NULL) == -1) {
        perror("sigaction for SIGTERM in parent");
        return 1;
    }

    std::cout << "Parent process running. Press Ctrl+C to send SIGINT or use `kill` to send SIGTERM." << std::endl;

    // Create child processes
    create_processes(pipefd);

    // Parent writes messages to children
    for (int i = 0; i < NUM_CHILDREN; ++i) {
        std::string message = "Hello from parent to child " + std::to_string(i);
        ssize_t bytesWritten = write(pipefd[i][1], message.c_str(), message.length() + 1);
        if (bytesWritten == -1) {
            perror("write");
            return 1;
        }
        close(pipefd[i][1]);  // Close the write end of the pipe
    }
    // Wait for all child processes to finish
    for (int i = 0; i < NUM_CHILDREN; ++i) {
        wait(nullptr);
    }

    std::cout << "Parent exiting..." << std::endl;
    return 0;
}
```

C++ Tab Width: 8 Ln 1, Col 1 INS

Execution :

```
rps@rps-virtual-machine:~/IPC_message$vim signal_pipe_ex.cpp
rps@rps-virtual-machine:~/IPC_message$rm signal_pipe_ex.cpp
rps@rps-virtual-machine:~/IPC_message$vim signal_pipe_ex.cpp
rps@rps-virtual-machine:~/IPC_message$make signal_pipe_ex
g++     signal_pipe_ex.cpp   -o signal_pipe_ex
rps@rps-virtual-machine:~/IPC_message$./signal_pipe_ex
Parent process running. Press Ctrl+C to send SIGINT or use `kill` to send SIGTERM.
Child 1 received message: Hello from parent to child 1
Child 0 received message: Hello from parent to child 0
Child 2 received message: Hello from parent to child 2
^CReceived SIGINT (Ctrl+C). Preparing to exit...
Received SIGINT (Ctrl+C). Preparing to exit...
Child 2 exiting...
Received SIGINT (Ctrl+C). Preparing to exit...
Child 1 exiting...
Parent exiting...
Received SIGINT (Ctrl+C). Preparing to exit...
Child 0 exiting...
rps@rps-virtual-machine:~/IPC_message$./signal_pipe_ex
Parent process running. Press Ctrl+C to send SIGINT or use `kill` to send SIGTERM.
Child 0 received message: Hello from parent to child 0
Child 2 received message: Hello from parent to child 2
Child 1 received message: Hello from parent to child 1
^Z
[21]+  Stopped                 ./signal_pipe_ex
rps@rps-virtual-machine:~/IPC_message$
```

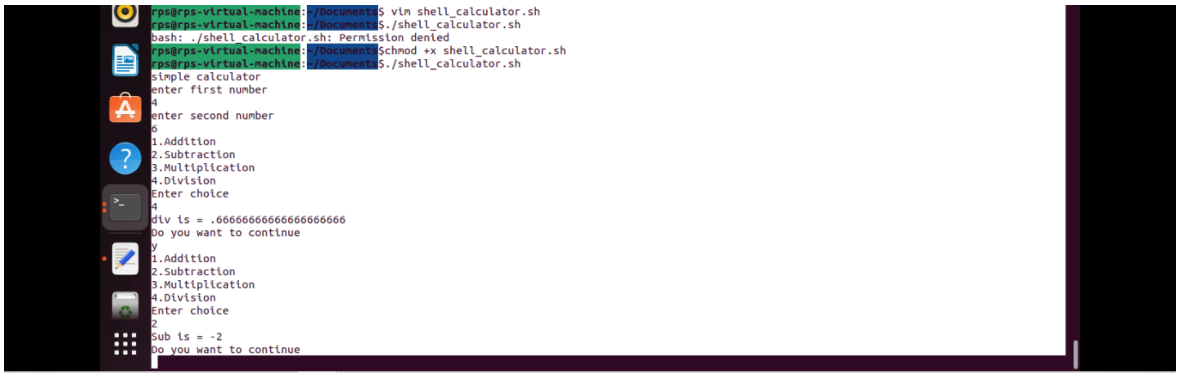
2. Shell Scripting

1. Example of Shell - based calculator.

Activities Text Editor Jul 31 12:06

Open shell_calculator.sh ~/Documents Save

```bash
#!/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
```

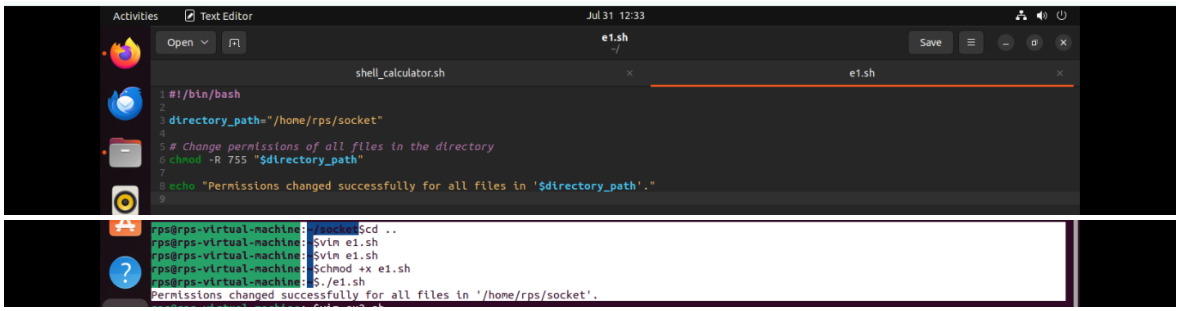


2. **Change File Permissions**

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.



3. **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

4. **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.

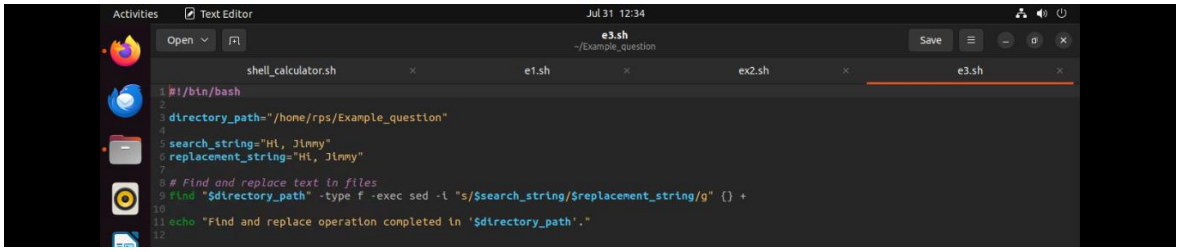
   Search for the specified string in all files within the directory.

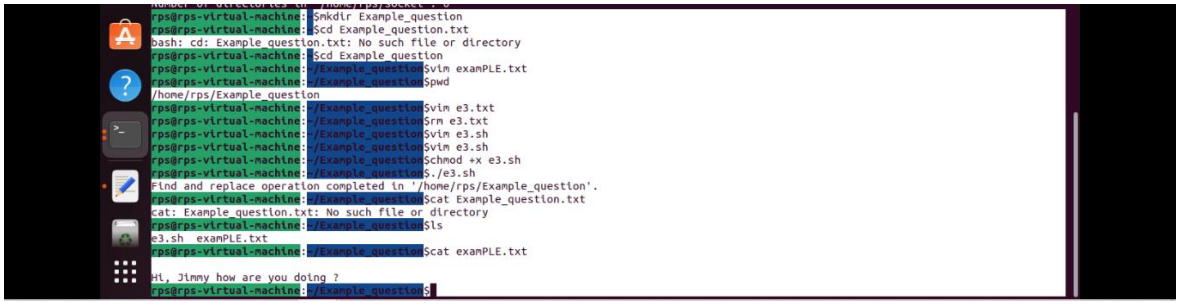
   Replace the string with the given replacement string in all occurrences.

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

   Sample Input:

   ./find_replace.sh /path/to/directory "old_text" "new_text"



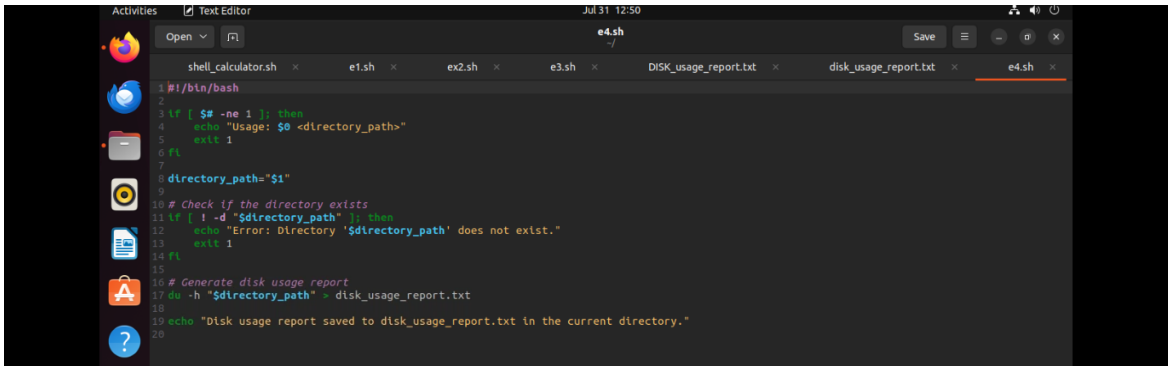


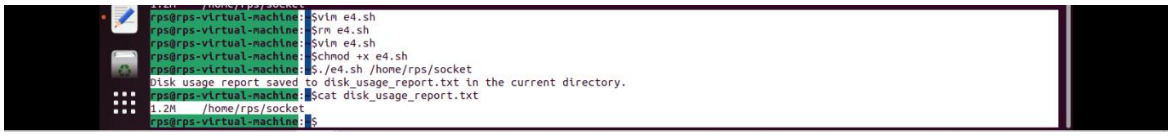
5. **Problem 4: Disk Usage Report**

   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.





6. **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:

   Instructions:

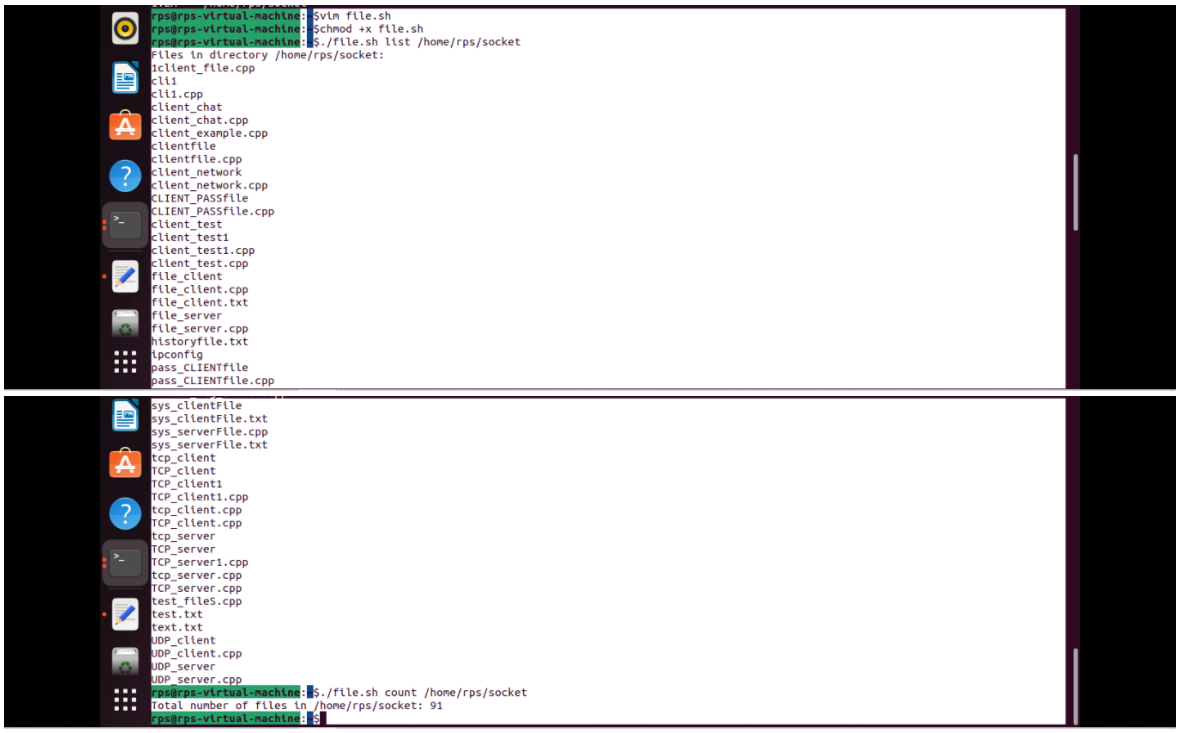
   List all files in the directory.
   Display the total number of files.
   Copy a specified file to a new location.
   Move a specified file to a new location.
   Delete a specified file.

Activities Text Editor Jul 31 14:15

Open file.sh ~/ Save

shell_calculator.sh e1.sh ex2.sh e3.sh DISK_usage_report.txt disk_usage_report.txt e4.sh file.sh

```bash
#!/bin/bash

# Function to list all files in the specified directory
list_files() {
    local dir=$1
    if [[ -d "$dir" ]]; then
        echo "Files in directory $dir:"
        ls -p "$dir" | grep -v /
    else
        echo "Directory $dir does not exist."
    fi
}

# Function to display the total number of files in the specified directory
count_files() {
    local dir=$1
    if [[ -d "$dir" ]]; then
        local count=$(ls -p "$dir" | grep -v / | wc -l)
        echo "Total number of files in $dir: $count"
    else
        echo "Directory $dir does not exist."
    fi
}

# Function to copy a specified file to a new location
copy_file() {
    local src_file=$1
    local dest_dir=$2
    if [[ -f "$src_file" ]] && [[ -d "$dest_dir" ]]; then
        cp "$src_file" "$dest_dir"
        echo "Copied $src_file to $dest_dir."
    else
        echo "Either source file $src_file or destination directory $dest_dir does not exist."
    fi
}

# Function to move a specified file to a new location
move_file() {
    local src_file=$1
    local dest_dir=$2
    if [[ -f "$src_file" ]] && [[ -d "$dest_dir" ]]; then
        mv "$src_file" "$dest_dir"
        echo "Moved $src_file to $dest_dir."
    else
        echo "Either source file $src_file or destination directory $dest_dir does not exist."
    fi
}

# Function to delete a specified file
delete_file() {
    local file=$1
    if [[ -f "$file" ]]; then
        rm "$file"
        echo "Deleted file $file."
    else
        echo "File $file does not exist."
    fi
}

# Main script logic to parse arguments
if [[ $# -lt 2 ]]; then
    echo "Usage: $0 <command> <arguments>"
    echo "Commands:"
    echo "  list <directory>"
    echo "  count <directory>"
    echo "  copy <source_file> <destination_directory>"
    echo "  move <source_file> <destination_directory>"
    echo "  delete <file>"
    exit 1
fi

command=$1
shift

case $command in
    list)
        list_files "$@"
        ;;
    count)
        count_files "$@"
        ;;
    copy)
        copy_file "$@"
        ;;
    move)
        move_file "$@"
        ;;
    delete)
        delete_file "$@"
        ;;
    *)
        echo "Unknown command: $command"
        echo "Usage: $0 <command> <arguments>"
        exit 1
        ;;
esac
```



7. **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:

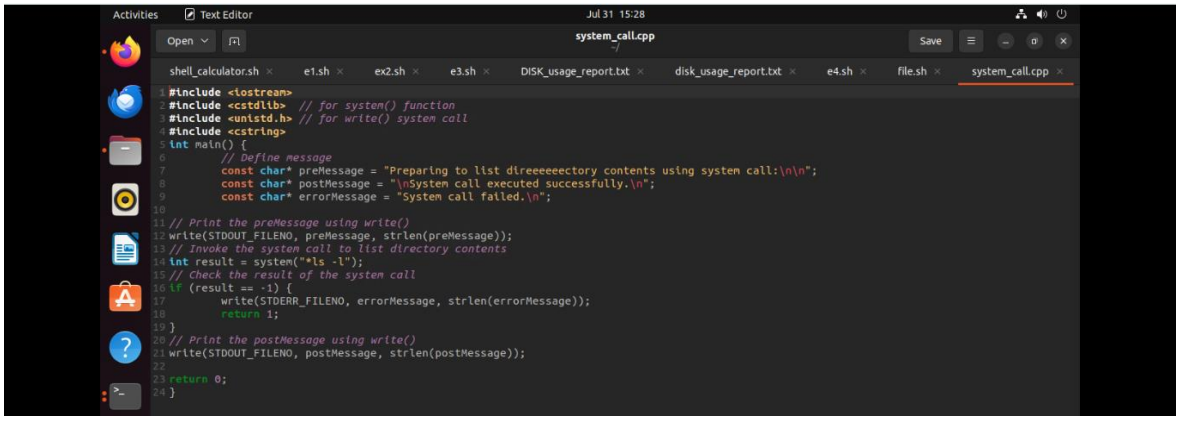
   List all files in the directory.

   Display the total number of files.

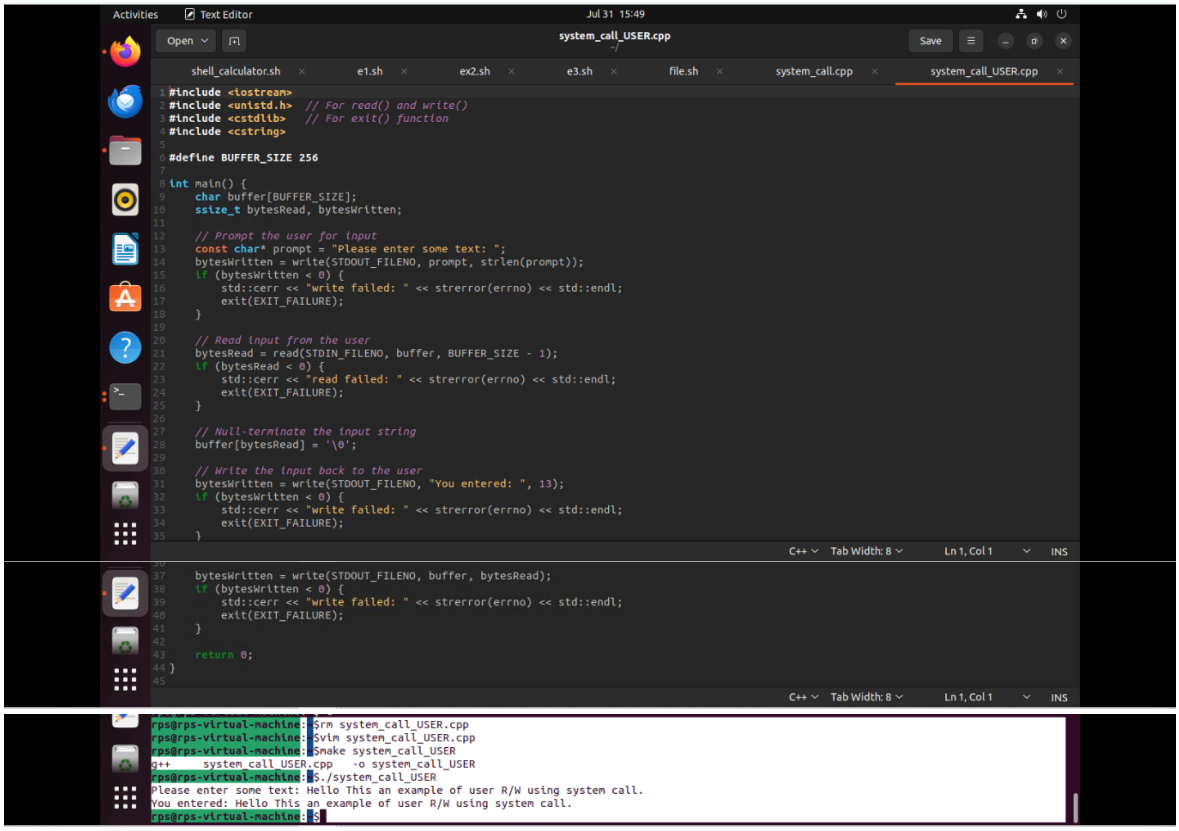
   Copy a specified file to a new location.

   Move a specified file to a new location.

   Delete a specified file.



8. **Problem Statement:** Write the code please read from user and write on screen using read and write API's in cpp using system calls.



9. **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:

   a. **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.

b. 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.

c. Read from the File:

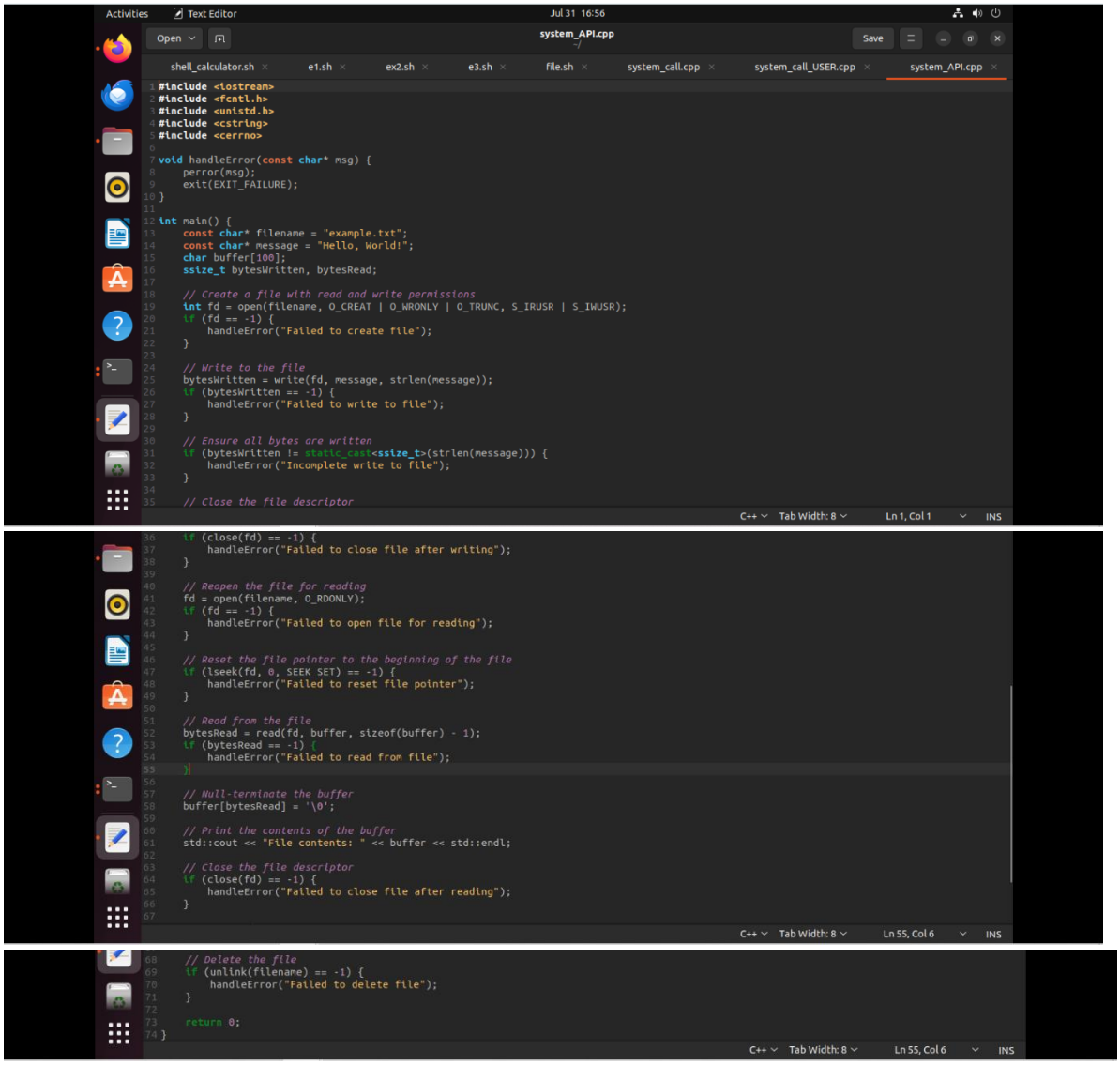
Use the lseek system call to reset the file pointer to the beginning of the file.
Read the contents of the file using the read system call and store it in a buffer.
Print the contents of the buffer to the standard output.

d. Delete the File:

Close the file descriptor using the close system call.
Use the unlink system call to delete the file "example.txt".

e. 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.



```cpp
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <cstring>
#include <cerrno>

void handleError(const char* msg) {
    perror(msg);
    exit(EXIT_FAILURE);
}

int main() {
    const char* filename = "example.txt";
    const char* message = "Hello, World!";
    char buffer[100];
    ssize_t bytesWritten, bytesRead;

    // Create a file with read and write permissions
    int fd = open(filename, O_CREAT | O_WRONLY | O_TRUNC, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        handleError("Failed to create file");
    }

    // Write to the file
    bytesWritten = write(fd, message, strlen(message));
    if (bytesWritten == -1) {
        handleError("Failed to write to file");
    }

    // Ensure all bytes are written
    if (bytesWritten != static_cast<ssize_t>(strlen(message))) {
        handleError("Incomplete write to file");
    }

    // Close the file descriptor
    if (close(fd) == -1) {
        handleError("Failed to close file after writing");
    }

    // Reopen the file for reading
    fd = open(filename, O_RDONLY);
    if (fd == -1) {
        handleError("Failed to open file for reading");
    }

    // Reset the file pointer to the beginning of the file
    if (lseek(fd, 0, SEEK_SET) == -1) {
        handleError("Failed to reset file pointer");
    }

    // Read from the file
    bytesRead = read(fd, buffer, sizeof(buffer) - 1);
    if (bytesRead == -1) {
        handleError("Failed to read from file");
    }

    // Null-terminate the buffer
    buffer[bytesRead] = '\0';

    // Print the contents of the buffer
    std::cout << "File contents: " << buffer << std::endl;

    // Close the file descriptor
    if (close(fd) == -1) {
        handleError("Failed to close file after reading");
    }

    // Delete the file
    if (unlink(filename) == -1) {
        handleError("Failed to delete file");
    }

    return 0;
}
```

```
rps@rps-virtual-machine:~$vim system_API.cpp
rps@rps-virtual-machine:~$make system_API
g++      system_API.cpp   -o system_API
rps@rps-virtual-machine:~$./system_API
File contents: Hello, World!
rps@rps-virtual-machine:~$
```