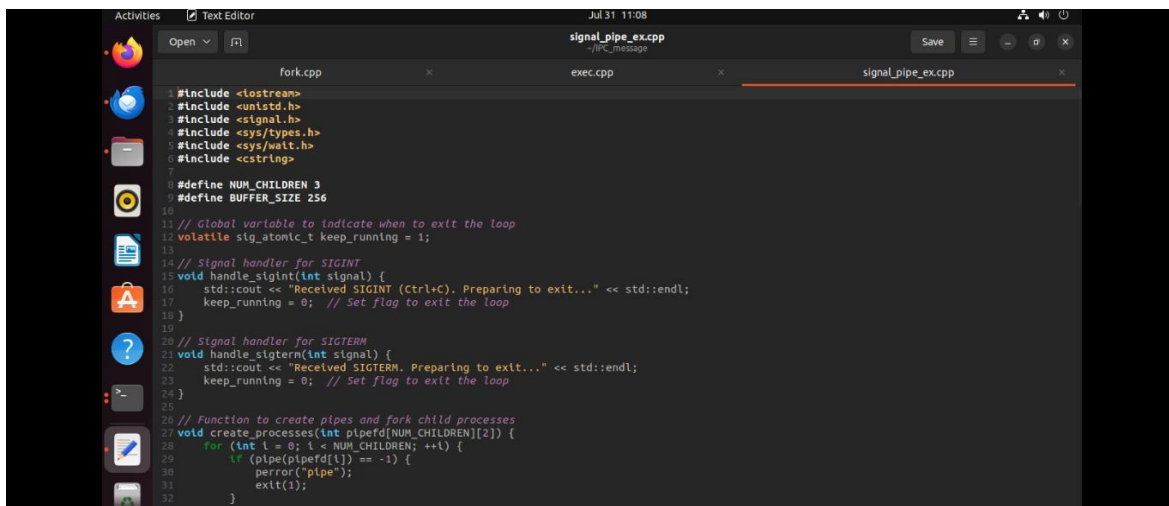


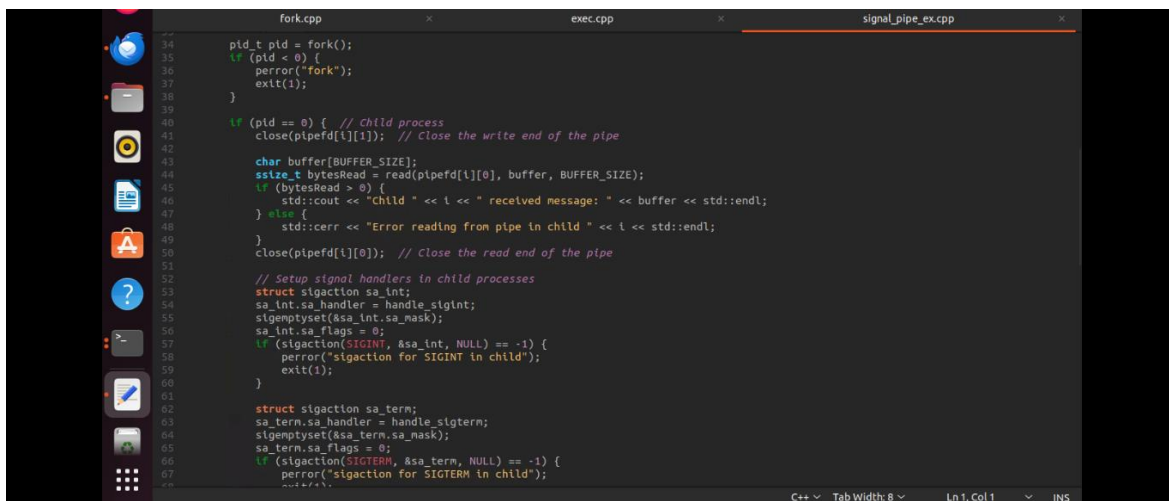
Day -12 LSP Assignment (Task – 1)

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

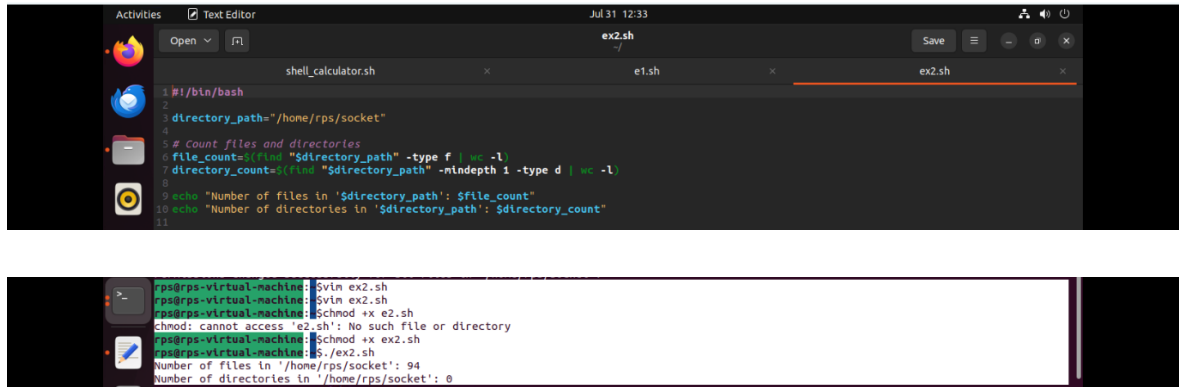
Design and implement a robust system in C++ that effectively utilizes signals to control the behavior of multiple processes and employs pipes for inter-process communication, enabling coordinated data exchange and process synchronization.



```
1 #include <iostream>
2 #include <unistd.h>
3 #include <signal.h>
4 #include <sys/types.h>
5 #include <sys/wait.h>
6 #include <cstring>
7
8 #define NUM_CHILDREN 3
9 #define BUFFER_SIZE 256
10
11 // Global variable to indicate when to exit the loop
12 volatile sig_atomic_t keep_running = 1;
13
14 // Signal handler for SIGINT
15 void handle_sigint(int signal) {
16     std::cout << "Received SIGINT (Ctrl+C). Preparing to exit..." << std::endl;
17     keep_running = 0; // Set flag to exit the loop
18 }
19
20 // Signal handler for SIGTERM
21 void handle_sigterm(int signal) {
22     std::cout << "Received SIGTERM. Preparing to exit..." << std::endl;
23     keep_running = 0; // Set flag to exit the loop
24 }
25
26 // Function to create pipes and fork child processes
27 void create_processes(int pipefd[NUM_CHILDREN][2]) {
28     for (int i = 0; i < NUM_CHILDREN; ++i) {
29         if (pipe(pipefd[i]) == -1) {
30             perror("pipe");
31             exit(1);
32         }
33     }
34 }
```



```
34 pid_t pid = fork();
35 if (pid < 0) {
36     perror("fork");
37     exit(1);
38 }
39
40 if (pid == 0) { // child process
41     close(pipefd[1][1]); // Close the write end of the pipe
42
43     char buffer[BUFFER_SIZE];
44     ssize_t bytesRead = read(pipefd[1][0], buffer, BUFFER_SIZE);
45     if (bytesRead > 0) {
46         std::cout << "Child " << i << " received message: " << buffer << std::endl;
47     } else {
48         std::cerr << "Error reading from pipe in child " << i << std::endl;
49     }
50     close(pipefd[1][0]); // Close the read end of the pipe
51
52     // Setup signal handlers in child processes
53     struct sigaction sa_int;
54     sa_int.sa_handler = handle_sigint;
55     sigemptyset(&sa_int.sa_mask);
56     sa_int.sa_flags = 0;
57     if (sigaction(SIGINT, &sa_int, NULL) == -1) {
58         perror("sigaction for SIGINT in child");
59         exit(1);
60     }
61
62     struct sigaction sa_term;
63     sa_term.sa_handler = handle_sigterm;
64     sigemptyset(&sa_term.sa_mask);
65     sa_term.sa_flags = 0;
66     if (sigaction(SIGTERM, &sa_term, NULL) == -1) {
67         perror("sigaction for SIGTERM in child");
68     }
69 }
```

```
#!/bin/bash
1
2 directory_path="/home/rps/socket"
3
4 # Count files and directories
5 file_count=$(find "$directory_path" -type f | wc -l)
6 directory_count=$(find "$directory_path" -mindepth 1 -type d | wc -l)
7
8 echo "Number of files in '$directory_path': $file_count"
9 echo "Number of directories in '$directory_path': $directory_count"
10
11
```

```
rps@rps-virtual-machine:~$ vim ex2.sh
rps@rps-virtual-machine:~$ vim ex2.sh
rps@rps-virtual-machine:~$ chmod +x ex2.sh
rps@rps-virtual-machine:~$ ./ex2.sh
Number of files in '/home/rps/socket': 94
Number of directories in '/home/rps/socket': 0
```

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.

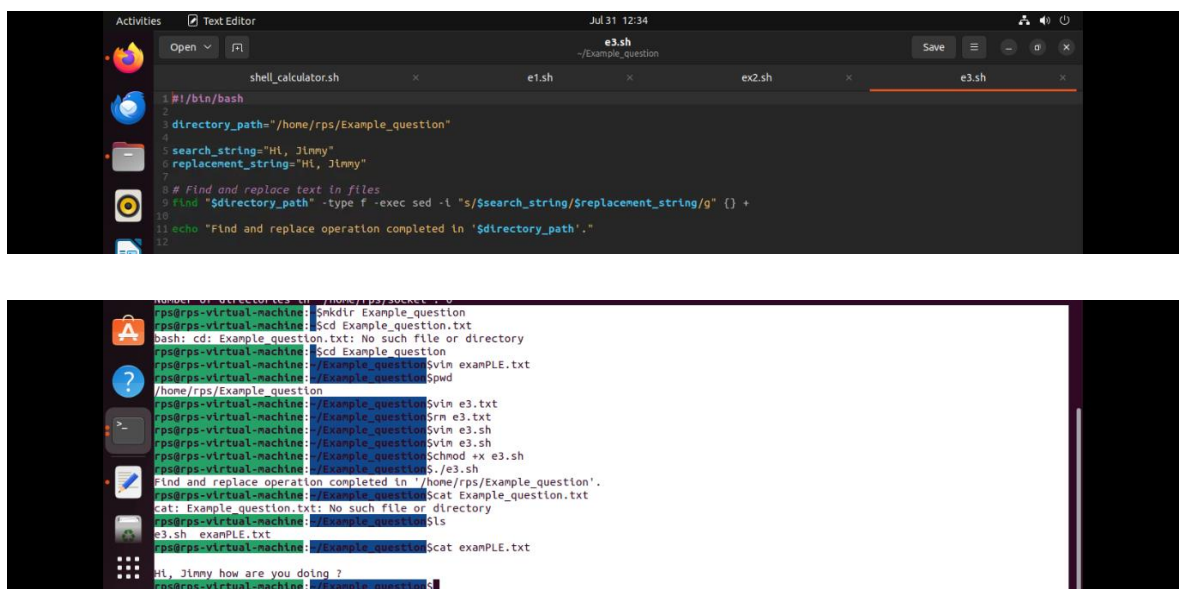
Search for the specified string in all files within the directory.

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"`



```
#!/bin/bash
1
2 directory_path="/home/rps/Example_question"
3
4 search_string="Hi, Jinny"
5 replacement_string="Hi, Jinny how are you doing?"
6
7 # Find and replace text in files
8 find "$directory_path" -type f -exec sed -i "s/$search_string/$replacement_string/g" {} +
9
10 echo "Find and replace operation completed in '$directory_path'."
11
12
```

```
rps@rps-virtual-machine:~$ mkdir Example_question
rps@rps-virtual-machine:~$ cd Example_question.txt
bash: cd: Example_question.txt: No such file or directory
rps@rps-virtual-machine:~$ cd Example_question
rps@rps-virtual-machine:~/Example_question$ vim exanPLE.txt
rps@rps-virtual-machine:~/Example_question$ pwd
/home/rps/Example_question
rps@rps-virtual-machine:~/Example_question$ vim e3.txt
rps@rps-virtual-machine:~/Example_question$ rm e3.txt
rps@rps-virtual-machine:~/Example_question$ vim e3.sh
rps@rps-virtual-machine:~/Example_question$ chmod +x e3.sh
rps@rps-virtual-machine:~/Example_question$ ./e3.sh
Find and replace operation completed in '/home/rps/Example_question'.
rps@rps-virtual-machine:~/Example_question$ cat Example_question.txt
cat: Example_question.txt: No such file or directory
rps@rps-virtual-machine:~/Example_question$ ls
e3.sh exanPLE.txt
rps@rps-virtual-machine:~/Example_question$ cat exanPLE.txt
Hi, Jinny how are you doing ?
rps@rps-virtual-machine:~/Example_question$
```

5. Problem 4: Disk Usage Report

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

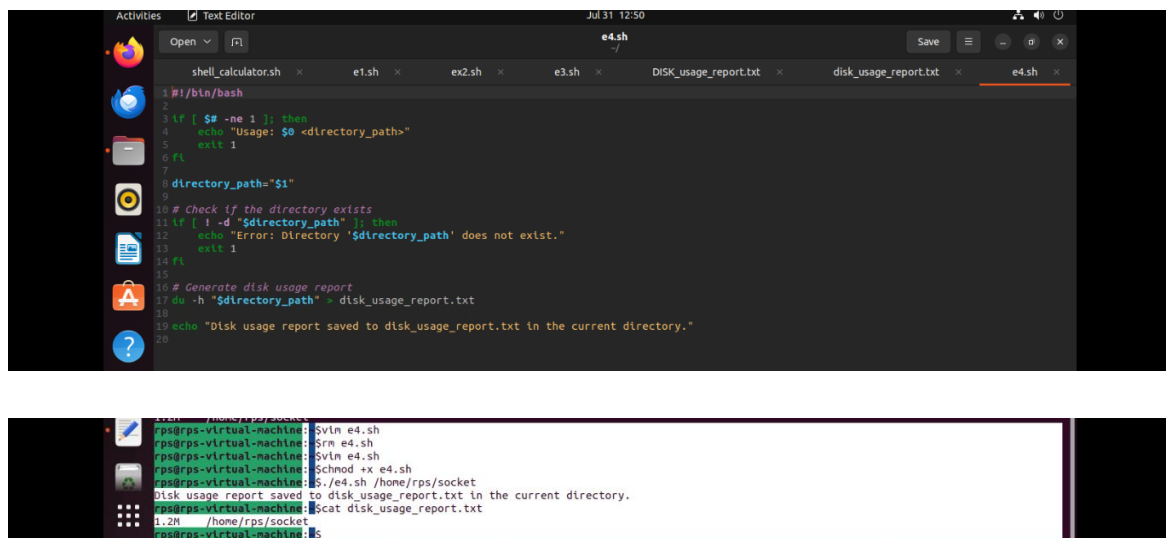
Instructions:

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.



```
Activities Text Editor Jul 31 12:50
e4.sh
Save
Open
shell_calculator.sh x e1.sh x ex2.sh x e3.sh x DISK_usage_report.txt x disk_usage_report.txt x e4.sh x
1 #!/bin/bash
2
3 if [ $# -ne 1 ]; then
4     echo "Usage: $0 <directory_path>"
5     exit 1
6 fi
7
8 directory_path="$1"
9
10 # Check if the directory exists
11 if [ ! -d "$directory_path" ]; then
12     echo "Error: Directory '$directory_path' does not exist."
13     exit 1
14 fi
15
16 # Generate disk usage report
17 du -h "$directory_path" > disk_usage_report.txt
18
19 echo "Disk usage report saved to disk_usage_report.txt in the current directory."
20
rpsdrps-virtual-machine: $ ./e4.sh /home/rps/socket
Disk usage report saved to disk_usage_report.txt in the current directory.
rpsdrps-virtual-machine: $ cat disk_usage_report.txt
1.2K /home/rps/socket
rpsdrps-virtual-machine: $
```

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

1 #!/bin/bash
2
3 # Function to list all files in the specified directory
4 list_files() {
5     local dir=$1
6     if [[ -d "$dir" ]]; then
7         echo "Files in directory $dir:"
8         ls -p "$dir" | grep -v /
9     else
10        echo "Directory $dir does not exist."
11    fi
12}
13
14 # Function to display the total number of files in the specified directory
15 count_files() {
16     local dir=$1
17     if [[ -d "$dir" ]]; then
18         local count=$(ls -p "$dir" | grep -v / | wc -l)
19         echo "Total number of files in $dir: $count"
20     else
21         echo "Directory $dir does not exist."
22     fi
23}
24
25 # Function to copy a specified file to a new location
26 copy_file() {
27     local src_file=$1
28     local dest_dir=$2
29     if [[ -f "$src_file" ]] && [[ -d "$dest_dir" ]]; then
30         cp "$src_file" "$dest_dir"
31         echo "Copied $src_file to $dest_dir."
32     else
33         echo "Either source file $src_file or destination directory $dest_dir does not exist."
34     fi
35}
36
37 # Function to move a specified file to a new location
38 move_file() {
39     local src_file=$1
40     local dest_dir=$2
41     if [[ -f "$src_file" ]] && [[ -d "$dest_dir" ]]; then
42         mv "$src_file" "$dest_dir"
43         echo "Moved $src_file to $dest_dir."
44     else
45         echo "Either source file $src_file or destination directory $dest_dir does not exist."
46     fi
47}
48
49 # Function to delete a specified file
50 delete_file() {
51     local file=$1
52     if [[ -f "$file" ]]; then
53         rm "$file"
54         echo "Deleted file $file."
55     else
56         echo "File $file does not exist."
57     fi
58}
59
60 # Main script logic to parse arguments
61 if [[ $# -lt 2 ]]; then
62     echo "Usage: $0 <command> <arguments>"
63
64     echo "Commands:"
65     echo "  list <directory>"
66     echo "  count <directory>"
67     echo "  copy <source_file> <destination_directory>"
68     echo "  move <source_file> <destination_directory>"
69     echo "  delete <file>"
70     exit 1
71 fi
72
73 command=$1
74 shift
75
76 case $command in
77     list)
78         list_files "$@"
79     ;;
80     count)
81         count_files "$@"
82     ;;
83     copy)
84         copy_file "$@"
85     ;;
86     move)
87         move_file "$@"
88     ;;
89     delete)
90         delete_file "$@"
91     ;;
92     *)
93         echo "Unknown command: $command"
94         echo "Usage: $0 <command> <arguments>"
95         exit 1
96     esac
97
```

```
rps@rps-virtual-machine:~$ gvim file.sh
rps@rps-virtual-machine:~$ ./file.sh
rps@rps-virtual-machine:~$ ./file.sh list /home/rps/socket
Files in directory /home/rps/socket:
client_file.cpp
cli1
cli1.cpp
client_chat
client_chat.cpp
client_example.cpp
clientFile
clientFile.cpp
client_network
client_network.cpp
CLIENT_PASSFile
CLIENT_PASSFile.cpp
client_test
client_test1
client_test1.cpp
client_test.cpp
file_client
file_client.cpp
file_client.txt
file_server
file_server.cpp
historyFile.txt
tpconfig
pass_CLIENTFile
pass_CLIENTFile.cpp

rps@rps-virtual-machine:~$ ./file.sh count /home/rps/socket
Total number of files in /home/rps/socket: 91
rps@rps-virtual-machine:~$
```

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.

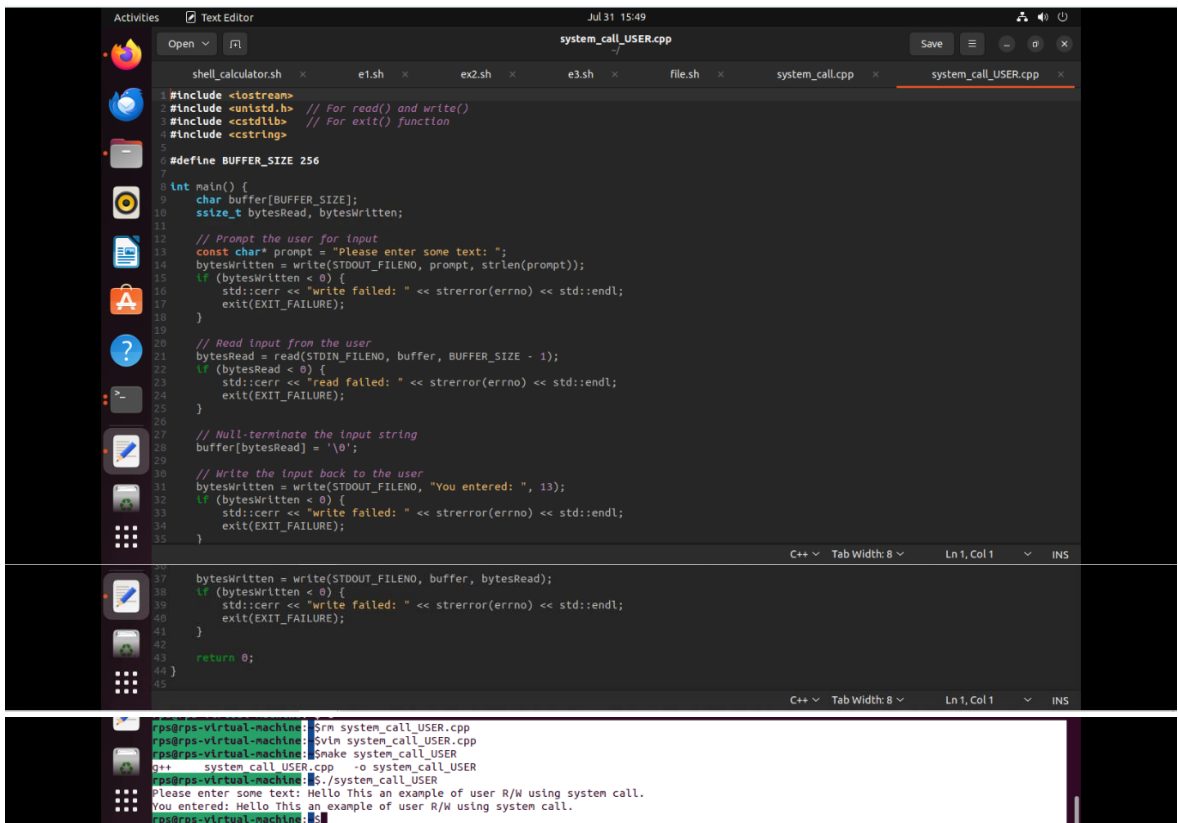
Delete a specified file.

```
Activities Text Editor Jul 31 15:28
system_call.cpp
Save
shell_calculator.sh x e1.sh x ex2.sh x e3.sh x DISK_usage_report.txt x disk_usage_report.txt x e4.sh x file.sh x system_call.cpp x
1 #include <iostream>
2 #include <cstdlib> // for system() function
3 #include <unistd.h> // for write() system call
4 #include <cstring>
5 int main() {
6     // Define message
7     const char* preMessage = "Preparing to list directory contents using system call:\n\n";
8     const char* postMessage = "\nSystem call executed successfully.\n";
9     const char* errorMessage = "System call failed.\n";
10
11 // Print the preMessage using write()
12 write(STDOUT_FILENO, preMessage, strlen(preMessage));
13 // Invoke the system call to list directory contents
14 int result = system("ls -l");
15 // Check the result of the system call
16 if (result == -1) {
17     write(STDERR_FILENO, errorMessage, strlen(errorMessage));
18     return 1;
19 }
20 // Print the postMessage using write()
21 write(STDOUT_FILENO, postMessage, strlen(postMessage));
22
23 return 0;
24 }
```



```
rps@rps-virtual-machine:~$ g++ system_call.cpp
rps@rps-virtual-machine:~$ ./system_call
Preparing to list directory contents using system call:
sh: 1: *ls: not found
System call executed successfully.
rps@rps-virtual-machine:~$
```

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.



```
system_call_USER.cpp
1 #include <iostream>
2 #include <unistd.h> // For read() and write()
3 #include <cstdlib> // For exit() function
4 #include <string>
5
6 #define BUFFER_SIZE 256
7
8 int main() {
9     char buffer[BUFFER_SIZE];
10    ssize_t bytesRead, bytesWritten;
11
12    // Prompt the user for input
13    const char prompt = "Please enter some text: ";
14    bytesWritten = write(STDOUT_FILENO, prompt, strlen(prompt));
15    if (bytesWritten < 0) {
16        std::cerr << "write failed: " << strerror(errno) << std::endl;
17        exit(EXIT_FAILURE);
18    }
19
20    // Read input from the user
21    bytesRead = read(STDIN_FILENO, buffer, BUFFER_SIZE - 1);
22    if (bytesRead < 0) {
23        std::cerr << "read failed: " << strerror(errno) << std::endl;
24        exit(EXIT_FAILURE);
25    }
26
27    // Null-terminate the input string
28    buffer[bytesRead] = '\0';
29
30    // Write the input back to the user
31    bytesWritten = write(STDOUT_FILENO, "You entered: ", 13);
32    if (bytesWritten < 0) {
33        std::cerr << "write failed: " << strerror(errno) << std::endl;
34        exit(EXIT_FAILURE);
35    }
36
37    bytesWritten = write(STDOUT_FILENO, buffer, bytesRead);
38    if (bytesWritten < 0) {
39        std::cerr << "write failed: " << strerror(errno) << std::endl;
40        exit(EXIT_FAILURE);
41    }
42
43    return 0;
44 }
45
```

```
rps@rps-virtual-machine:~$ rm system_call_USER.cpp
rps@rps-virtual-machine:~$ g++ system_call_USER.cpp
rps@rps-virtual-machine:~$ ./system_call_USER
Please enter some text: Hello This an example of user R/W using system call.
You entered: Hello This an example of user R/W using system call.
rps@rps-virtual-machine:~$
```

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.

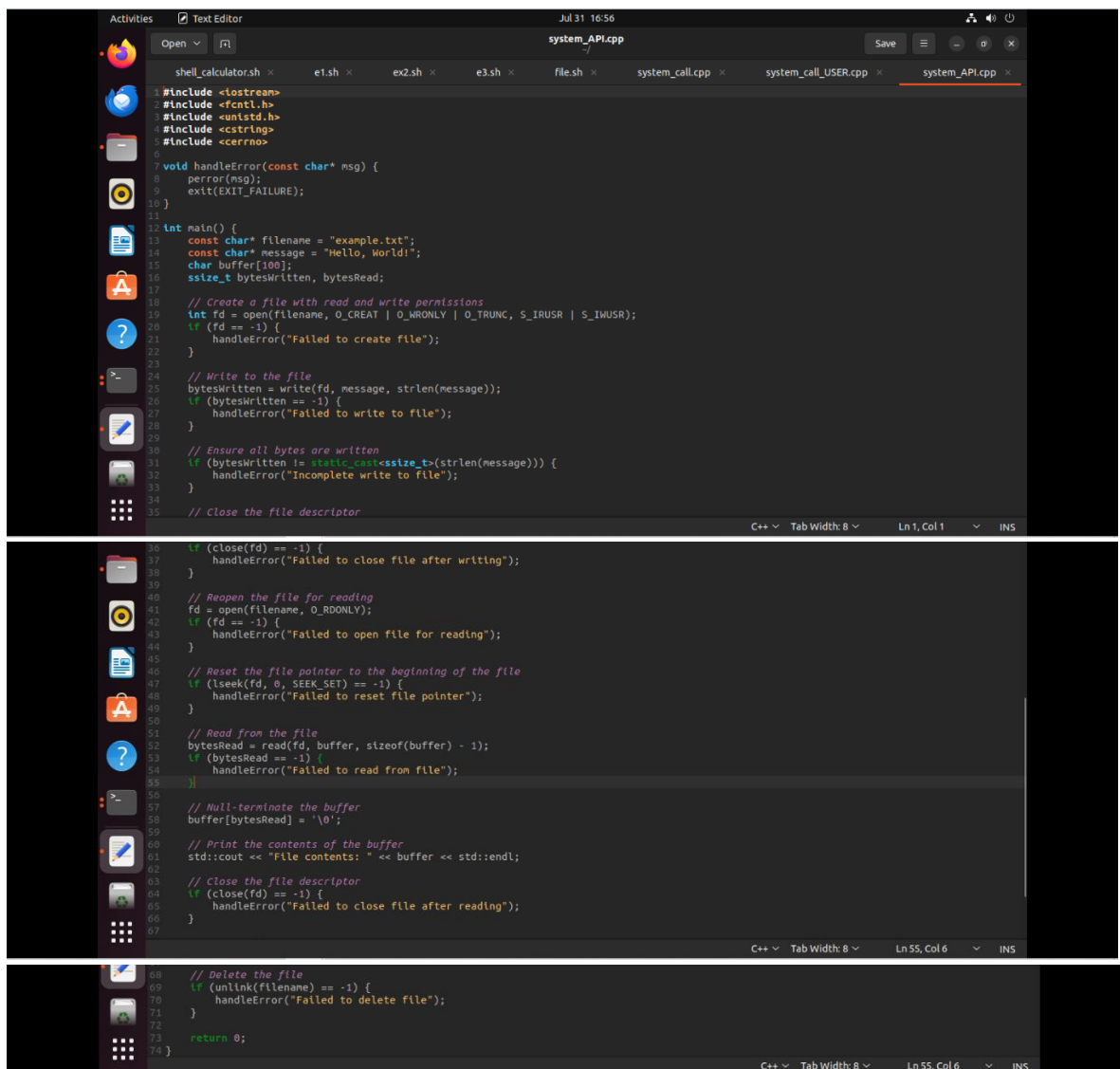
d. Delete the File:

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.



```
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <string>
#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 != 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;
}
```

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