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.

```
rps@rps-virtual-machine:~/pipe$ vim ipc.cpp
rps@rps-virtual-machine:~/pipe$ make ipc
g++     ipc.cpp   -o ipc
rps@rps-virtual-machine:~/pipe$ ./ipc
Parent: Enter a message (or 'exit' to terminate): hii hello child
Parent: Enter a message (or 'exit' to terminate): Child: Received message: hii hello child
ok bye
Parent: Enter a message (or 'exit' to terminate): Child: Received message: ok bye
exit
Parent: Terminating child process and exiting...
rps@rps-virtual-machine:~/pipe$
```

```cpp
#include <iostream>
#include <unistd.h>
#include <csignal>
#include <cstring>
#include <sys/wait.h>

using namespace std;

int pipefd[2];
pid_t childPid;

void signalHandler(int sig_num) {
    // Close write end of the pipe
    close(pipefd[1]);
    // Send termination signal to child process
    kill(childPid, SIGTERM);
    // Wait for the child process to terminate
    wait(NULL);
    cout << "\nParent: Received SIGINT. Terminating child process and cleaning up...\n";
    exit(0);
}

void childProcess() {
    close(pipefd[1]); // Close write end of the pipe in the child

    char buffer[100];
    while (true) {
        memset(buffer, 0, sizeof(buffer));
        ssize_t bytesRead = read(pipefd[0], buffer, sizeof(buffer) - 1);
        if (bytesRead > 0) {
            cout << "Child: Received message: " << buffer << endl;
        } else {
            break; // If no bytes read, break the loop
        }
    }
```

```
void childProcess() {
    close(pipefd[1]); // Close write end of the pipe in the child

    char buffer[100];
    while (true) {
        memset(buffer, 0, sizeof(buffer));
        ssize_t bytesRead = read(pipefd[0], buffer, sizeof(buffer) - 1);
        if (bytesRead > 0) {
            cout << "Child: Received message: " << buffer << endl;
        } else {
            break; // If no bytes read, break the loop
        }
    }

    close(pipefd[0]); // Close read end of the pipe
    cout << "Child: Terminating...\n";
}

void parentProcess() {
    close(pipefd[0]); // Close read end of the pipe in the parent

    string message;
    while (true) {
        cout << "Parent: Enter a message (or 'exit' to terminate): ";
        getline(cin, message);

        if (message == "exit") {
            // Close the write end of the pipe
            close(pipefd[1]);
            // Send termination signal to child process
            kill(childPid, SIGTERM);
```

```
            break;
        } else {
            write(pipefd[1], message.c_str(), message.size());
        }
    }
}

int main() {
    // Setup signal handler for SIGINT
    signal(SIGINT, sigintHandler);

    if (pipe(pipefd) == -1) {
        cerr << "Pipe creation failed!\n";
        return 1;
    }

    childPid = fork();
    if (childPid == -1) {
        cerr << "Fork failed!\n";
        return 1;
    }

    if (childPid == 0) {
        // In child process
        childProcess();
    } else {
        // In parent process
        parentProcess();
    }

    return 0;
}
```

ECHO COMMANDS :-

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.

```
rps@rps-virtual-machine:~/pipe$ vim change_directory.sh
rps@rps-virtual-machine:~/pipe$ chmod +x change_directory.sh
rps@rps-virtual-machine:~/pipe$ ./change_directory.sh ~pipe
Error: ~pipe is not a directory
rps@rps-virtual-machine:~/pipe$ ./change_directory.sh ~/pipe
Permissions of all files in /home/rps/pipe have been changed to rwxr-xr-x
rps@rps-virtual-machine:~/pipe$ ls -l ~/pipe
total 152
drwxr-xr-x 2 rps rps  4096 Jul 29 17:01 amee
-rwxr-xr-x 1 rps rps   489 Jul 31 12:58 change_directory.sh
-rwxr-xr-x 1 rps rps 18248 Jul 31 09:17 ipc
-rwxr-xr-x 1 rps rps  2186 Jul 31 10:13 ipc.cpp
-rwxr-xr-x 1 rps rps 17800 Jul 30 09:20 ipc_example
-rwxr-xr-x 1 rps rps  4749 Jul 30 12:08 ipc_example.cpp
-rwxr-xr-x 1 rps rps 84064 Jul 30 16:44 pipeline
-rwxr-xr-x 1 rps rps  3934 Jul 30 16:44 pipeline.cpp
-rwxr-xr-x 1 rps rps  1506 Jul 29 16:59 pipe_send.cpp
rps@rps-virtual-machine:~/pipe$
```

```
#!/bin/bash

# Check if exactly one argument is given
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <directory-path>"
    exit 1
fi

# Get the directory path from the argument
DIR=$1

# Check if the given argument is a directory
if [ ! -d "$DIR" ]; then
    echo "Error: $DIR is not a directory"
    exit 1
fi

# Change permissions of all files in the specified directory
chmod -R 755 "$DIR"

# Print completion message
echo "Permissions of all files in $DIR have been changed to rwxr-xr-x"
~
~
```

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

```
rps@rps-virtual-machine:~/linux31task$ vim count_files_dir.sh
rps@rps-virtual-machine:~/linux31task$ chmod +x count_files_dir.sh
```

```
rps@rps-virtual-machine:~/linux31task$ chmod +x count_files_dir.sh
rps@rps-virtual-machine:~/linux31task$ ./count_files_dir.sh /home/rps/linux31task
Number of files: 7
Number of directories: 1
rps@rps-virtual-machine:~/linux31task$
```

```
#!/bin/bash

# Check if exactly one argument is given
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <directory-path>"
    exit 1
fi

# Get the directory path from the argument
DIR=$1

# Check if the given argument is a directory
if [ ! -d "$DIR" ]; then
    echo "Error: $DIR is not a directory or does not exist"
    exit 1
fi

# Count the number of files and directories
num_files=$(find "$DIR" -maxdepth 1 -type f | wc -l)
num_dirs=$(find "$DIR" -maxdepth 1 -type d | wc -l)

# Adjust the directory count to exclude the parent directory itself
num_dirs=$((num_dirs - 1))

# Print the counts with appropriate labels
echo "Number of files: $num_files"
echo "Number of directories: $num_dirs"
~
~
```

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"

```
rps@rps-virtual-machine:~/linux31task$ vim find_replace.sh
rps@rps-virtual-machine:~/linux31task$ chmod +x find_replace.sh
rps@rps-virtual-machine:~/linux31task$ ls
calculator.sh  change_directory.sh  count_files_dir.sh  example_dir  find_replace.sh  ipc.cpp  quotess  quotes.sh  quotess.sh
```

```
rps@rps-virtual-machine:~/linux31task$ echo > "THIS IS OLD TEXT ."">file1.txt
rps@rps-virtual-machine:~/linux31task$ echo "This is my old text."">file.txt
rps@rps-virtual-machine:~/linux31task$ echo "this is my another instant old text."">file2.txt
rps@rps-virtual-machine:~/linux31task$ chmod +x find_replace.sh
rps@rps-virtual-machine:~/linux31task$ ./find_replace.sh /home/rps/linux31task "old" "new"
Completed find and replace operation in /home/rps/linux31task
rps@rps-virtual-machine:~/linux31task$
```

```bash
#!/bin/bash

# Check if exactly three arguments are given
if [ "$#" -ne 3 ]; then
    echo "Usage: $0 <directory-path> <search-string> <replacement-string>"
    exit 1
fi

# Get the directory path, search string, and replacement string from the arguments
DIR=$1
SEARCH_STRING=$2
REPLACEMENT_STRING=$3

# Check if the given argument is a directory
if [ ! -d "$DIR" ]; then
    echo "Error: $DIR is not a directory or does not exist"
    exit 1
fi

# Find and replace the string in all files within the directory
find "$DIR" -type f -exec sed -i "s/$SEARCH_STRING/$REPLACEMENT_STRING/g" {} +

# Print a completion message
echo "Completed find and replace operation in $DIR"
~
~
~
~
~
~
```

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

```
rps@rps-virtual-machine:~/linux31task$ vim disk_usage.sh
rps@rps-virtual-machine:~/linux31task$ chmod +x disk_usage.sh
rps@rps-virtual-machine:~/linux31task$ ./disk_usage.sh linux31task
Error: linux31task is not a directory or does not exist
rps@rps-virtual-machine:~/linux31task$ ./disk_usage.sh /home/rps/linux31task
Disk usage report saved to /home/rps/linux31task/disk_usage_report.txt
rps@rps-virtual-machine:~/linux31task$
```

```
#!/bin/bash

# Check if exactly one argument is given
if [ "$#" -ne 1 ]; then
    echo "Usage: $0 <directory-path>"
    exit 1
fi

# Get the directory path from the argument
DIR=$1

# Check if the given argument is a directory
if [ ! -d "$DIR" ]; then
    echo "Error: $DIR is not a directory or does not exist"
    exit 1
fi

# Generate the disk usage report and save it to disk_usage_report.txt
du -sh "$DIR" > disk_usage_report.txt

# Print a completion message
echo "Disk usage report saved to $(pwd)/disk_usage_report.txt"
~
```

PROGRAM FOR SIMPLE CALCULATION :-

```
#!/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 "Subtraction is =" $sum;;
3)sum=$(echo "$n1 * $n2" | bc -l)
echo "Multiplication is =" $sum;;
4)sum=$(echo "$n1 / $n2" | bc -l)
echo "Division is =" $sum;;
*)echo "invalid choice"
esac
echo "Do you want to continue"
read i
if [ $i != "y" ]
then
exit
fi
done
"calculator.sh" 35L, 621B
```

```
rps@rps-virtual-machine:~/linux31task$ vim calculator.sh
rps@rps-virtual-machine:~/linux31task$ chmod +x calculator.sh
rps@rps-virtual-machine:~/linux31task$ ./calculator.sh
simple calculator
enter first number
17
enter second number
39
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
1
Addition is = 56
Do you want to continue
2
rps@rps-virtual-machine:~/linux31task$ ./calculator
bash: ./calculator: No such file or directory
rps@rps-virtual-machine:~/linux31task$ ./calculator.sh
simple calculator
enter first number
17
enter second number
39
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
```

```
enter second number
39
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
1
Addition is = 56
Do you want to continue
y
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
2
Subtraction is = -22
Do you want to continue
y
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
3
Multiplication is = 663
Do you want to continue
y
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
4
```

```
Do you want to continue
y
1.Addition
2.Subtraction
3.Multiplication
4.Division
Enter choice
4
Division is = .43589743589743589743
Do you want to continue
n
rps@rps-virtual-machine:~/linux31task$
```

```
rps@rps-virtual-machine:~/linux31task$ ls -l calculator.sh
-rwxrwxr-x 1 rps rps 621 Jul 31 12:13 calculator.sh
rps@rps-virtual-machine:~/linux31task$
```

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.

```
rps@rps-virtual-machine:~/linux31task$ vim file_management.sh
rps@rps-virtual-machine:~/linux31task$ chmod +x file_management.sh
rps@rps-virtual-machine:~/linux31task$ ./file_management.sh /home/rps/linux31task list
Files in /home/rps/linux31task:
calculator.sh
change_directory.sh
count_files_dir.sh
disk_usage_report.txt
disk_usage.sh
example_dir
file1.txt
file2.txt
file_management.sh
file.txt
find_replace.sh
ipc.cpp
quotess
quotes.sh
quotess.sh
'THIS IS OLD TEXT .'
rps@rps-virtual-machine:~/linux31task$ ./file_management.sh /home/rps/linux31task count
Total number of files in /home/rps/linux31task: 16
rps@rps-virtual-machine:~/linux31task$
```

```
rps@rps-virtual-machine:~/linux31task$ vim file_management.sh
rps@rps-virtual-machine:~/linux31task$ mkdir backup
rps@rps-virtual-machine:~/linux31task$ ls
 backup                count_files_dir.sh      example_dir    file_management.sh   ipc.cpp     quotess.sh
 calculator.sh         disk_usage_report.txt   file1.txt      file.txt             quotess     'THIS IS OLD TEXT .'
 change_directory.sh   disk_usage.sh           file2.txt      find_replace.sh      quotes.sh
rps@rps-virtual-machine:~/linux31task$ ./file_management.sh /home/rps/linux31task copy file1.txt backup
Copied file1.txt to backup
rps@rps-virtual-machine:~/linux31task$ ./file_management.sh /home/rps/linux31task move file2.txt backup
Moved file2.txt to backup
rps@rps-virtual-machine:~/linux31task$ ./file_management.sh /home/rps/linux31task delete file1.txt
Deleted file1.txt
rps@rps-virtual-machine:~/linux31task$
```

```bash
#!/bin/bash

# Function to list all files in the directory
list_files() {
    echo "Files in $DIR:"
    ls -1 "$DIR"
}

# Function to display the total number of files in the directory
count_files() {
    count=$(ls -1 "$DIR" | wc -l)
    echo "Total number of files in $DIR: $count"
}

# Function to copy a specified file to a new location
copy_file() {
    if [ -f "$DIR/$1" ]; then
        cp "$DIR/$1" "$2"
        echo "Copied $1 to $2"
    else
        echo "File $1 does not exist in $DIR"
    fi
}

# Function to move a specified file to a new location
move_file() {
    if [ -f "$DIR/$1" ]; then
        mv "$DIR/$1" "$2"
        echo "Moved $1 to $2"
    else
        echo "File $1 does not exist in $DIR"
    fi
}

# Function to delete a specified file
```

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

# Check if at least one argument is given
if [ "$#" -lt 1 ]; then
    echo "Usage: $0 <directory-path> [function] [args...]"
    exit 1
fi

# Get the directory path from the first argument
DIR=$1
shift

# Check if the given argument is a directory
if [ ! -d "$DIR" ]; then
    echo "Error: $DIR is not a directory or does not exist"
    exit 1
fi

# Execute the requested function
case $1 in
    list)
        list_files
        ;;
    count)
```

```
# Execute the requested function
case $1 in
    list)
        list_files
        ;;
    count)
        count_files
        ;;
    copy)
        if [ "$#" -ne 3 ]; then
            echo "Usage: $0 <directory-path> copy <source-file> <destination>"
            exit 1
        fi
        copy_file "$2" "$3"
        ;;
    move)
        if [ "$#" -ne 3 ]; then
            echo "Usage: $0 <directory-path> move <source-file> <destination>"
            exit 1
        fi
        move_file "$2" "$3"
        ;;
    delete)
        if [ "$#" -ne 2 ]; then
            echo "Usage: $0 <directory-path> delete <file>"
            exit 1
        fi
        delete_file "$2"
        ;;
    *)
        echo "Invalid function. Valid functions are: list, count, copy, move, delete"
        exit 1
        ;;
esac
```

CODE FOR SYSTEM CALL :-

```
rps@rps-virtual-machine:~/pipe$ vim system_call.cpp
rps@rps-virtual-machine:~/pipe$ make system_call
g++     system_call.cpp   -o system_call
rps@rps-virtual-machine:~/pipe$ ./system_call
Preparing to list directory contents using system call:

total 176
drwxr-xr-x 2 rps rps  4096 Jul 29 17:01 amee
-rwxr-xr-x 1 rps rps   489 Jul 31 13:03 change_directory.sh
-rwxr-xr-x 1 rps rps   489 Jul 31 13:08 count_files_directory.sh
-rwxr-xr-x 1 rps rps 18248 Jul 31 09:17 ipc
-rwxr-xr-x 1 rps rps  2186 Jul 31 10:13 ipc.cpp
-rwxr-xr-x 1 rps rps 17800 Jul 30 09:20 ipc_example
-rwxr-xr-x 1 rps rps  4749 Jul 30 12:08 ipc_example.cpp
-rwxr-xr-x 1 rps rps 84064 Jul 30 16:44 pipeline
-rwxr-xr-x 1 rps rps  3934 Jul 30 16:44 pipeline.cpp
-rwxr-xr-x 1 rps rps  1506 Jul 29 16:59 pipe_send.cpp
-rwxrwxr-x 1 rps rps 16376 Jul 31 15:35 system_call
-rw-rw-r-- 1 rps rps   558 Jul 31 15:34 system_call.cpp

System call executed successfully.
rps@rps-virtual-machine:~/pipe$
```

```cpp
#include <iostream>
#include <cstdlib>
#include <unistd.h>
#include <cstring>

int main() {
        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";

        write(STDOUT_FILENO, preMessage, strlen(preMessage));

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

        if(result == -1) {
                write(STDERR_FILENO, errorMessage, strlen(errorMessage));
                return 1;
        }

        write(STDOUT_FILENO, postMessage, strlen(postMessage));
        return 0;
}
```

PROGRAM FOR SYSTEM CALL(READ AND WRITE) :-

```
rps@rps-virtual-machine:~/pipe$ vim systemcall_rw.cpp
rps@rps-virtual-machine:~/pipe$ make systemcall_rw
g++     systemcall_rw.cpp    -o systemcall_rw
rps@rps-virtual-machine:~/pipe$ ./systemcall_rw
Preparing to list directory contents using system call.
total 200
drwxr-xr-x 2 rps rps  4096 Jul 29 17:01 amee
-rwxr-xr-x 1 rps rps   489 Jul 31 13:03 change_directory.sh
-rwxr-xr-x 1 rps rps   489 Jul 31 13:08 count_files_directory.sh
-rwxr-xr-x 1 rps rps 18248 Jul 31 09:17 ipc
-rwxr-xr-x 1 rps rps  2186 Jul 31 10:13 ipc.cpp
-rwxr-xr-x 1 rps rps 17800 Jul 30 09:20 ipc_example
-rwxr-xr-x 1 rps rps  4749 Jul 30 12:08 ipc_example.cpp
-rwxr-xr-x 1 rps rps 84064 Jul 30 16:44 pipeline
-rwxr-xr-x 1 rps rps  3934 Jul 30 16:44 pipeline.cpp
-rwxr-xr-x 1 rps rps  1506 Jul 29 16:59 pipe_send.cpp
-rwxrwxr-x 1 rps rps 16376 Jul 31 15:35 system_call
-rw-rw-r-- 1 rps rps   558 Jul 31 15:40 system_call.cpp
-rwxrwxr-x 1 rps rps 16472 Jul 31 15:49 systemcall_rw
-rw-rw-r-- 1 rps rps  1512 Jul 31 15:49 systemcall_rw.cpp

System call executed successfully.
Please enter a message: hii ameesha
You entered: hii ameesha
```

```cpp
#include <iostream>
#include <cstdlib> // For system() function
#include <unistd.h> // For write() and read() system calls
#include <cstring>

int main() {
    // Define messages
    const char* preMessage = "Preparing to list directory contents using system call.\n";
    const char* postMessage = "\nSystem call executed successfully.\n";
    const char* errorMessage = "System call failed.\n";
    const char* userPrompt = "Please enter a message: ";
    char userInput[256];

    // 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));

    // Prompt the user for input
    write(STDOUT_FILENO, userPrompt, strlen(userPrompt));

    // Read the user input
    ssize_t bytesRead = read(STDIN_FILENO, userInput, sizeof(userInput) - 1);

    if (bytesRead > 0) {
```

```
    // Print the postMessage using write()
    write(STDOUT_FILENO, postMessage, strlen(postMessage));

    // Prompt the user for input
    write(STDOUT_FILENO, userPrompt, strlen(userPrompt));

    // Read the user input
    ssize_t bytesRead = read(STDIN_FILENO, userInput, sizeof(userInput) - 1);

    if (bytesRead > 0) {
        // Null-terminate the input to make it a valid C-string
        userInput[bytesRead] = '\0';

        // Write the user input to the screen
        write(STDOUT_FILENO, "You entered: ", 13);
        write(STDOUT_FILENO, userInput, bytesRead);
    } else {
        write(STDERR_FILENO, "Failed to read user input.\n", 27);
    }

    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.

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

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

void handleError(const char* msg) {
    std::cerr << msg << ": " << strerror(errno) << std::endl;
    exit(EXIT_FAILURE);
}

int main() {
    int fd;
    const char* filename = "example.txt";
    const char* message = "Hello, World!";
    char buffer[50];

    // Create or truncate the file
    fd = open(filename, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (fd == -1) {
        handleError("Error opening file");
    }

    // Write to the file
    ssize_t bytes_written = write(fd, message, strlen(message));
    if (bytes_written == -1) {
        handleError("Error writing to file");
    }

    // Ensure all bytes are written
    if (bytes_written != (ssize_t)strlen(message)) {
        handleError("Error: Not all bytes written to file");
    }

    // Reset file pointer to the beginning of the file
```

```
    // Ensure all bytes are written
    if (bytes_written != (ssize_t)strlen(message)) {
        handleError("Error: Not all bytes written to file");
    }

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

    // Read from the file
    ssize_t bytes_read = read(fd, buffer, sizeof(buffer) - 1);
    if (bytes_read == -1) {
        handleError("Error reading from file");
    }

    // Null-terminate the buffer to print as a string
    buffer[bytes_read] = '\0';
    std::cout << "File contents: " << buffer << std::endl;

    // Close the file
    if (close(fd) == -1) {
        handleError("Error closing file");
    }

    // Delete the file
    if (unlink(filename) == -1) {
        handleError("Error deleting file");
    }

    return 0;
}
```

Question:

Explain the role of virtual memory in Linux memory management. How does the kernel use system calls like brk, mmap, and munmap to manage virtual memory for processes? Discuss the implications of overcommitting memory and the mechanisms Linux employs to handle memory pressure.

Potential Areas for Further Exploration:

Deep dive into specific system calls: Explore the inner workings of brk, mmap, and munmap in detail, including their parameters, return values, and common use cases.

Memory allocation algorithms: Discuss different memory allocation strategies used by the kernel, such as the buddy system and slab allocator.

Performance implications: Analyze the performance impact of different memory management techniques under various workloads.

Memory management in specific scenarios: Explore memory management challenges and solutions in specific use cases like containerization or real-time systems.

Role of Virtual Memory in Linux Memory Management

Virtual memory in Linux is a memory management technique that provides an application with the illusion of a large, contiguous block of memory, regardless of the actual physical memory available. It allows multiple processes to run concurrently without interfering with each other, as each process

operates in its own isolated virtual address space.

Key Functions of Virtual Memory

Isolation and Protection: Each process operates in its own virtual address space, protecting it from other processes and the kernel. This isolation prevents processes from accessing each other's memory, which enhances security and stability.

Address Space Extension: Virtual memory allows applications to use more memory than is physically available by swapping pages of memory between physical RAM and disk storage.

Efficient Memory Usage: Virtual memory enables the kernel to optimize memory usage and handle memory fragmentation more effectively by mapping virtual addresses to physical addresses as needed.

Handling Memory Pressure

Linux employs several mechanisms to handle memory pressure:

Swapping: When physical memory is full, Linux swaps out inactive pages to disk to free up RAM. This process helps in managing memory pressure but can lead to performance degradation due to slower disk access.

Page Replacement Algorithms: The kernel uses algorithms like Least Recently Used (LRU) to manage which pages to swap out or keep in memory.

OOM Killer: When the system runs out of memory, the OOM Killer terminates processes to free up memory. The kernel selects processes based on various heuristics such as memory usage and process priority.

Memory Cgroups: In containerized environments, cgroups (control groups) are used to limit and prioritize memory usage per container. This prevents a single container from consuming excessive resources.

Further Exploration

1. System Call Internals:

brk and sbrk: Investigate how these system calls interact with the heap management in the kernel and libc.

mmap and munmap: Explore the implementation details in the kernel and the use of these calls for file-backed and anonymous mappings.

2. Memory Allocation Algorithms:

Buddy System: Used for physical memory management, splitting memory into partitions to manage

allocation and deallocation efficiently.

Slab Allocator: Used for cache management, optimizing the allocation of objects of the same size.

3. Performance Implications:

Analyze how different allocation strategies impact performance under various workloads, including high-concurrency scenarios and large-scale applications.

4. Special Scenarios:

Containerization: Examine how memory management strategies differ in containerized environments compared to traditional VMs.

Real-Time Systems: Explore how Linux handles memory management in real-time systems to meet strict timing constraints and avoid latency issues.

By delving into these areas, you can gain a deeper understanding of Linux's memory management strategies and their implications for system performance and reliability.