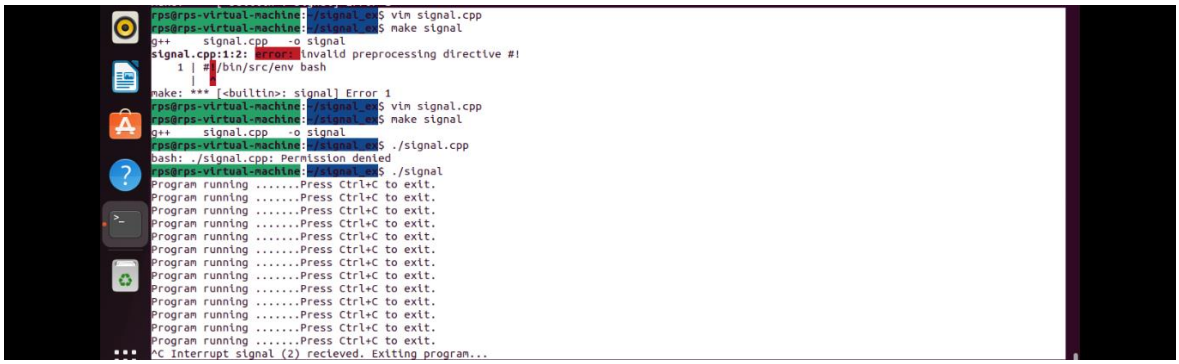
Day – 5 Assignment

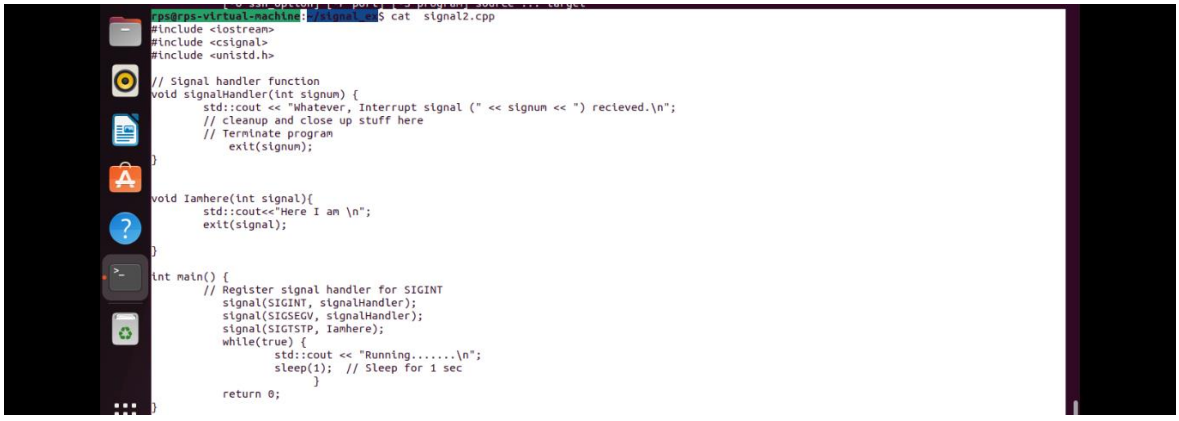
1. **Simple Signal Handler:** **Write a C++ program that handles the SIGINT signal (Ctrl+C) gracefully by printing a custom message before exiting.**

```
rps@rps-virtual-machine:~/signal_ex$ cat signal.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
    cout << " Interrupt signal (" << signum <<") recieved. Exiting program..."<<endl;
    exit(signum);

}

int main (){

signal(SIGINT, signalHandler);
while (true){
        cout << "Program running .......Press Ctrl+C to exit." <<endl;
        sleep(1);
      }
return 0;
}
rps@rps-virtual-machine:~/signal_ex$
```



```
rps@rps-virtual-machine:~/signal_ex$ vim signal.cpp
rps@rps-virtual-machine:~/signal_ex$ make signal
g++     signal.cpp   -o signal
signal.cpp:1:2: error: invalid preprocessing directive #!
    1 | #!/bin/src/env bash
      |  ^
make: *** [<builtin>: signal] Error 1
rps@rps-virtual-machine:~/signal_ex$ vim signal.cpp
rps@rps-virtual-machine:~/signal_ex$ make signal
g++     signal.cpp   -o signal
rps@rps-virtual-machine:~/signal_ex$ ./signal.cpp
bash: ./signal.cpp: Permission denied
rps@rps-virtual-machine:~/signal_ex$ ./signal
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
Program running .......Press Ctrl+C to exit.
^C Interrupt signal (2) recieved. Exiting program...
```

2. **Multiple Signal Handling:** **Create a program that handles both SIGINT and SIGTERM signals, printing a different message for each.**



```
rps@rps-virtual-machine:~/signal_ex$ cat  signal2.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function
void signalHandler(int signum) {
        std::cout << "Whatever, Interrupt signal (" << signum << ") recieved.\n";
        // cleanup and close up stuff here
        // Terminate program
            exit(signum);
}


void Iamhere(int signal){
        std::cout<<"Here I am \n";
        exit(signal);

}

int main() {
        // Register signal handler for SIGINT
           signal(SIGINT, signalHandler);
           signal(SIGSEGV, signalHandler);
           signal(SIGTSTP, Iamhere);
           while(true) {
                   std::cout << "Running.......\n";
                   sleep(1);  // Sleep for 1 sec
                         }
           return 0;
}
```

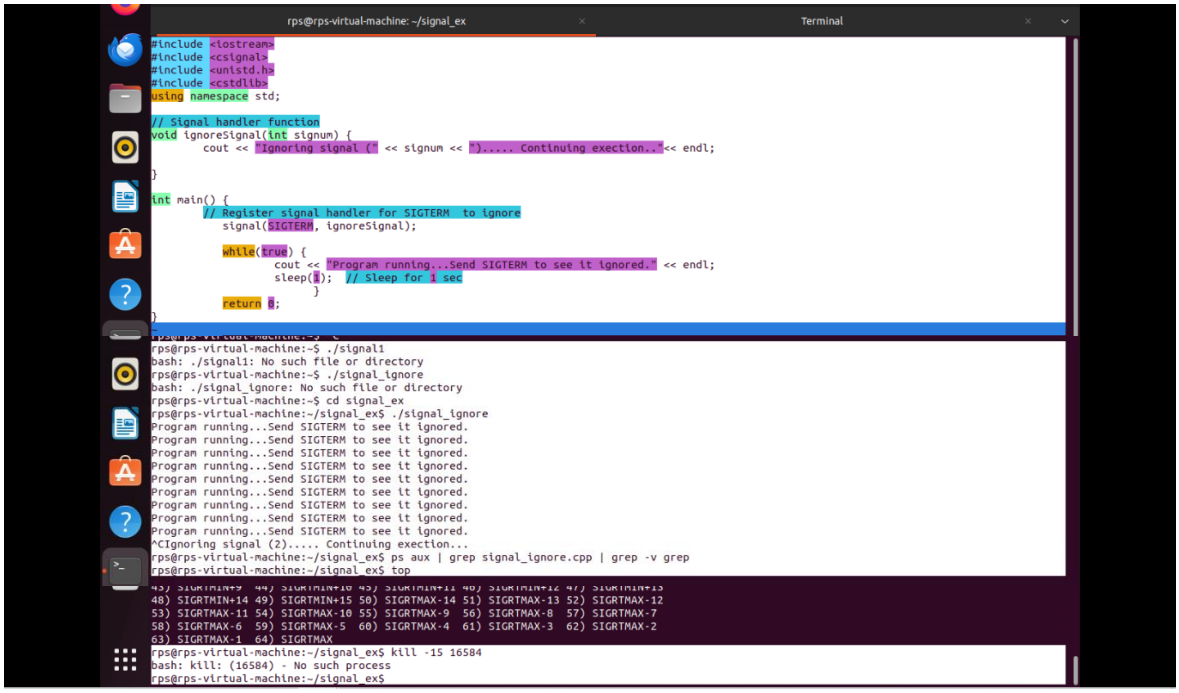
```
rps@rps-virtual-machine:~/signal_ex$  ./ signal2.cpp
bash: ./: Is a directory
rps@rps-virtual-machine:~/signal_ex$ ./signal2
Running.......
Running.......
Running.......
Running.......
Running.......
Running.......
Running.......
Running.......
Running.......
^CWhatever, Interrupt signal (2) recieved.
rps@rps-virtual-machine:~/signal_ex$ ./signal2
Running.......
Running.......
Running.......
Running.......
Running.......
^ZHere I am
rps@rps-virtual-machine:~/signal_ex$
```

Or we could use if – else block to achieve the code-exection.

```
rps@rps-virtual-machine:~/signal_ex$ cat mult_signalHandlers.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
        if (signum == SIGINT ) {
                cout << "Interrupt signal recieved....." <<endl;
        }
                        else if (signum == SIGTERM ) {
                                cout << "Terminate the signal recieved. Exiting the program........" <<endl;
                        }       else if(signum == SIGQUIT ) {
                                cout << "Quit the process...." <<endl;
                        }

exit(signum);
}

int main () {

signal(SIGINT, signalHandler);
signal(SIGTERM, signalHandler);
signal(SIGQUIT, signalHandler);


while(true) {
        cout << "Program running....Press CTRL+C to exit" <<endl;
sleep(1);
}
return 0;
}
rps@rps-virtual-machine:~/signal_ex$
```

```
rps@rps-virtual-machine:~/signal_ex$ vim mult_signalHandlers.cpp
rps@rps-virtual-machine:~/signal_ex$ vim mult_signalHandlers.cpp
rps@rps-virtual-machine:~/signal_ex$ make mult_signalHandlers
g++     mult_signalHandlers.cpp   -o mult_signalHandlers
rps@rps-virtual-machine:~/signal_ex$ ./mult_signalHandlers
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
^CInterrupt signal recieved.....
rps@rps-virtual-machine:~/signal_ex$ ./mult_signalHandlers
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
^Z
[4]+  Stopped                 ./mult_signalHandlers
rps@rps-virtual-machine:~/signal_ex$
rps@rps-virtual-machine:~/signal_ex$ ./mult_signalHandlers
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
Program running....Press CTRL+C to exit
^\Quit the process....
rps@rps-virtual-machine:~/signal_ex$
```

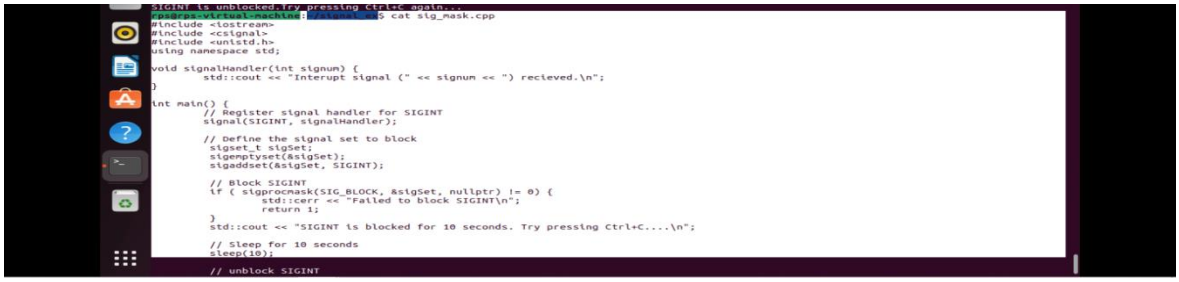
3. **Ignoring Signals: Develop a program that ignores the SIGTERM signal and continues execution even after it's sent.**

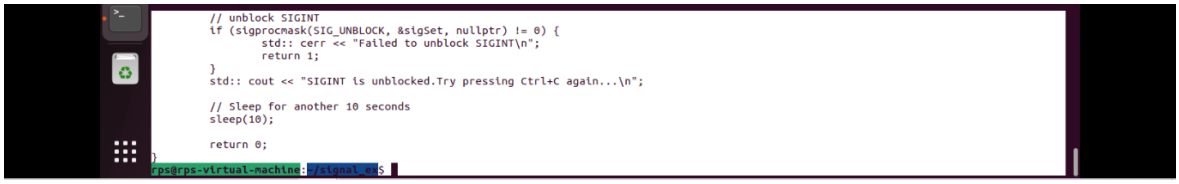


4. **sig_mask or signal masking (SIGINT)**

   a. Using **sigmask** typically refers to manipulating signal masks, which control the set of signals that a process can block from being delivered. In C and C++, this is usually done using the POSIX signal handling functions such as **sigprocmask, sigemptyset, sigfillset, sigaddset, and sigdelset.**

Here's an example program that demonstrates how to block and unblock signals using these functions. **The program will block SIGINT (Ctrl+C) for a period of time, then unblock it.**





```
rps@rps-virtual-machine:~/signal_ex$ vim sig_mask.cpp
rps@rps-virtual-machine:~/signal_ex$ make sig_mask
g++     sig_mask.cpp   -o sig_mask
rps@rps-virtual-machine:~/signal_ex$ ./sig_mask
SIGINT is blocked for 10 seconds. Try pressing Ctrl+C....
SIGINT is unblocked.Try pressing Ctrl+C again...
rps@rps-virtual-machine:~/signal_ex$
```

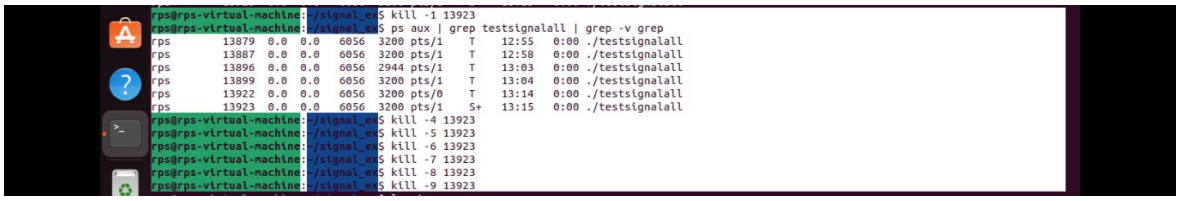
5. sig_mask1 (SIGTERM)

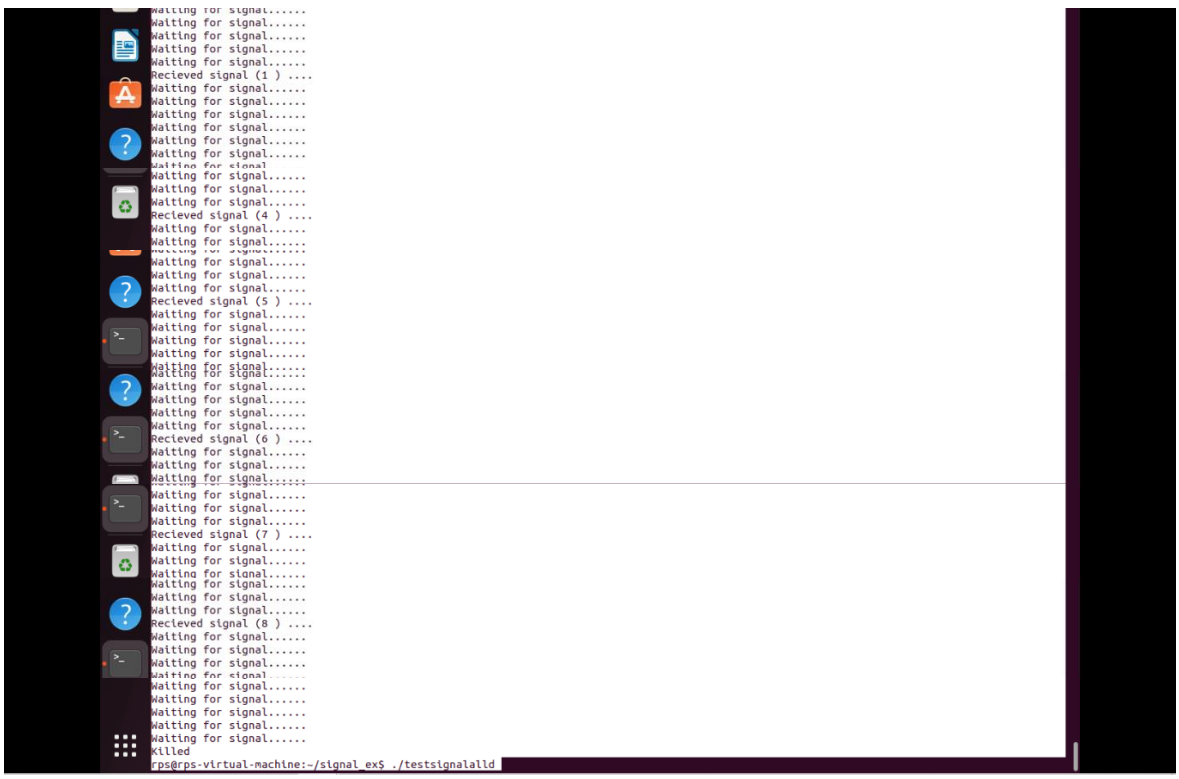
```
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
        std::cout << "Interupt signal (" << signum << ") recieved.\n";
}

int main() {
        // Register signal handler for SIGINT
        signal(SIGTERM, signalHandler);

        // Define the signal set to block
         sigset_t sigSet;
         sigemptyset(&sigSet);
         sigaddset(&sigSet, SIGTERM);

         // Block SIGINT
         if ( sigprocmask(SIG_BLOCK, &sigSet, nullptr) != 0) {
                std::cerr << "Failed to block SIGTERM\n";
                return 1;
         }
         std::cout << "SIGTERM is blocked during critical section.\n";

         // ritical section starts
         std::cout <<"Entering  Critical section...\n";
         sleep(5);   // Stimulating crtical section
         std::cout <<"Exiting critical section...\n";
         // Critical section ends

         // unblock SIGINT
         if (sigprocmask(SIG_UNBLOCK, &sigSet, nullptr) != 0) {
                std:: cerr << "Failed to unblock SIGTERM\n";
                return 1;
         }
        std:: cout << "SIGTERM is unblocked.\n";

        // Sleep to demonstrate signal handling after unblocking
        sleep(10);

        return 0;
}
rps@rps-virtual-machine:~/signal_ex$
```

```
rps@rps-virtual-machine:~/signal_ex$ ./sig_mask1
SIGTERM is blocked during critical section.
Entering        Critical section...
Exiting critical section...
SIGTERM is unblocked.
rps@rps-virtual-machine:~/signal_ex$
```

6. Program for ‘multiple signal’ call using single ‘signal Handler’.

```
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void myHandler(int signum) {

std::cout << "Recieved signal ("  << signum << " ) ....\n";

}

int main() {
        void(*pfRet)(int);
        pfRet = std::signal(SIGHUP, myHandler);  //1
        pfRet = std::signal(SIGINT, myHandler);  //2
        pfRet = std::signal(SIGQUIT, myHandler);  //3
        pfRet = std::signal(SIGILL, myHandler);   //4
        pfRet = std::signal(SIGTRAP, myHandler);  //5
        pfRet = std::signal(SIGABRT, myHandler);  //6
        pfRet = std::signal(SIGBUS, myHandler);   //7
        pfRet = std::signal(SIGFPE, myHandler);   //8
        pfRet = std::signal(SIGKILL, myHandler);  //9


        while(true){
                std::cout <<"Waiting for signal......"<< std::endl;
                sleep(1);
        }
        return 0;
}
"testsignalall.cpp" 31L, 779B                                   28,1-8        All
```





2. Assignment :

1. Problem Statement 1: Signal Masking and Unmasking for Graceful Shutdown

Problem: Develop a C++ application that gracefully handles termination signals (e.g., SIGTERM, SIGINT) by masking specific signals during critical operations and unmasking them afterwards. Implement a clean shutdown procedure that ensures all resources are released before the process exits.

Key Challenges:

Determining the appropriate signals to mask during critical operations. Ensuring timely unmasking of signals to avoid process hangs. Implementing a robust shutdown mechanism that handles unexpected interruptions.

a. SIGINT - Signal Masking and Unmasking for Graceful Shutdown.

```
SIGINT is unblocked.Try pressing Ctrl+C again...
rps@rps-virtual-machine:~/signal_ex$ cat sig_mask.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
        std::cout << "Interupt signal (" << signum << ") recieved.\n";
}

int main() {
        // Register signal handler for SIGINT
        signal(SIGINT, signalHandler);

        // Define the signal set to block
         sigset_t sigSet;
         sigemptyset(&sigSet);
         sigaddset(&sigSet, SIGINT);

         // Block SIGINT
         if ( sigprocmask(SIG_BLOCK, &sigSet, nullptr) != 0) {
                 std::cerr << "Failed to block SIGINT\n";
                 return 1;
         }
         std::cout << "SIGINT is blocked for 10 seconds. Try pressing Ctrl+C....\n";

         // Sleep for 10 seconds
         sleep(10);

         // unblock SIGINT
```

```
         // unblock SIGINT
         if (sigprocmask(SIG_UNBLOCK, &sigSet, nullptr) != 0) {
                 std:: cerr << "Failed to unblock SIGINT\n";
                 return 1;
         }
         std:: cout << "SIGINT is unblocked.Try pressing Ctrl+C again...\n";

         // Sleep for another 10 seconds
         sleep(10);

         return 0;
}
rps@rps-virtual-machine:~/signal_ex$
```

```
rps@rps-virtual-machine:~/signal_ex$ vim sig_mask.cpp
rps@rps-virtual-machine:~/signal_ex$ make sig_mask
g++      sig_mask.cpp    -o sig_mask
rps@rps-virtual-machine:~/signal_ex$ ./sig_mask
SIGINT is blocked for 10 seconds. Try pressing Ctrl+C....
SIGINT is unblocked.Try pressing Ctrl+C again...
rps@rps-virtual-machine:~/signal_ex$
```

b. SIGTERM - Signal Masking and Unmasking for Graceful Shutdown.

```
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
        std::cout << "Interupt signal (" << signum << ") recieved.\n";
}

int main() {
        // Register signal handler for SIGINT
        signal(SIGTERM, signalHandler);

        // Define the signal set to block
         sigset_t sigSet;
         sigemptyset(&sigSet);
         sigaddset(&sigSet, SIGTERM);

         // Block SIGINT
         if ( sigprocmask(SIG_BLOCK, &sigSet, nullptr) != 0) {
                 std::cerr << "Failed to block SIGTERM\n";
                 return 1;
         }
         std::cout << "SIGTERM is blocked during critical section.\n";

         // ritical section starts
         std::cout <<"Entering  Critical section...\n";
         sleep(5);   // Stimulating crtical section
         std::cout <<"Exiting critical section...\n";
         // Critical section ends

         // unblock SIGINT
         if (sigprocmask(SIG_UNBLOCK, &sigSet, nullptr) != 0) {
                 std:: cerr << "Failed to unblock SIGTERM\n";
                 return 1;
         }
```

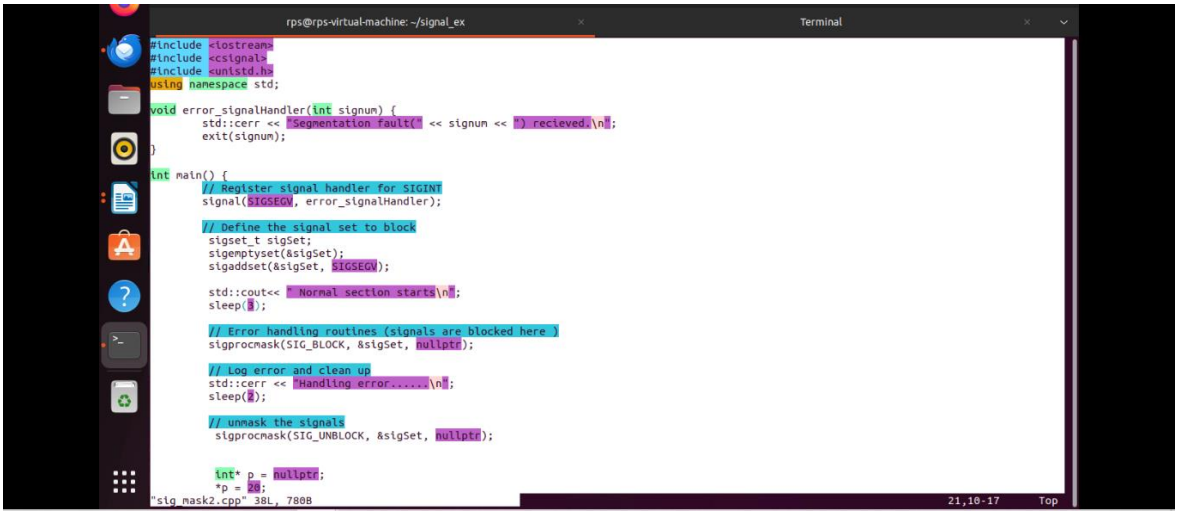
```
         std:: cout << "SIGTERM is unblocked.\n";

         // Sleep to demonstrate signal handling after unblocking
         sleep(10);

         return 0;
}
rps@rps-virtual-machine:~/signal_ex$
```

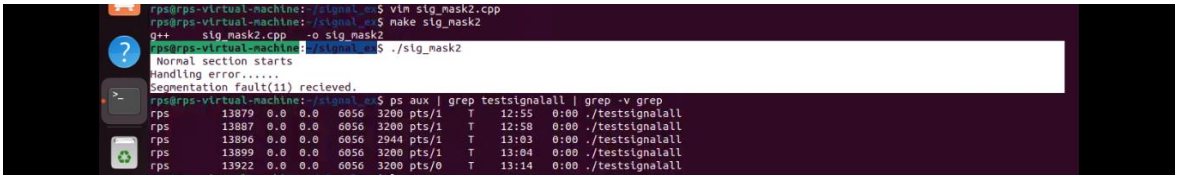
2. Problem Statement 2: Signal Masking and Unmasking for Error Handling

Problem: Create a C++ application that uses signal masking and unmasking to handle errors gracefully. Mask specific signals during error handling routines to prevent recursive signal delivery. Implement a mechanism to log error details and perform necessary cleanup actions before re-enabling the masked signals.

Key Challenges:

Identifying the appropriate signals to mask during error handling. Preventing infinite recursion of signal handlers. Ensuring proper error logging and resource cleanup.





3. Files Manipulation using signal Handlers

Execution :

```
#include <iostream>
#include <csignal>
#include <cstdlib>
#include <cstdio>
#include <chrono>
#include <thread>

//File name to clean up
const char* fileName = "tempfile.txt";

// Signal handler function
void signalHandler(int signum)
{
        std::cout << "\nInterrupt signal (" << signum << ") recieved.\n";

        // Cleanup and clean up stuff here
        if (std::remove(fileName) == 0) {
                std::cout << " File " << fileName << " deleted successfully.\n";
        } else {
                std::perror("Error deleting file");
        }
        // Terminate program
        exit(signum);
}

int main() {
        // Create a file to be monitored
        FILE* file = std::fopen(fileName, "w");
        if (file == nullptr) {
                std::perror("Error creating file");
                return 1;
        }
        std::fputs(" Temporary file content.", file);
        std::fclose(file);
        std::cout << " File " << fileName << " created.\n";
"signal_file_handling.cpp" 47L, 1097B                                   13,1          Top
```

```
        //Register signal handler for SIGINT
        std::signal(SIGINT, signalHandler);

        std::cout << "Press Ctrl+C to register the signal handler....\n";

        // Infinte loop
        while(true){
                std::this_thread::sleep_for(std::chrono::seconds(1));
        }
        return 0;
}
                                                                        47,2-8        Bot
```

4. Use of Default ‘ signal Handling ‘

```
rps@rps-virtual-machine: ~/signal_ex                    Terminal
#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function
void signalHandler(int signum) {
        std::cout << "Caught Signal " << signum << std::endl;
}

int main () {
        // Register signal handler for SIGINT
        signal(SIGINT, signalHandler);

        std::cout << "Press Ctrl+C ro generate SIGINT signal." << std::endl;
        std::cout <<"Waiting for signal....."<< std::endl;

        // Sleep to allow time for user to press Ctrl+C
        sleep(10);

        // Ignore SIGINT signal
        std::cout << "Ignoring SIGINT signal for the next 10 seconds." <<std::endl;
        signal(SIGINT, SIG_IGN);

        // Sleep to allow time for user to press Ctrl+C
        sleep(10);

        // Restore default handling for SIGINT
        // std::cout<< "Restoring default handling for SIGINT signal." << std::endl;
        signal(SIGINT, SIG_DFL);

        // Sleep for allow time for user to press Ctrl+C
        sleep(10);

        std::cout<< "Exiting program" << std::endl;
        return 0;
"sig_defaultHandler.cpp" 36L, 909B                                      3,18          Top
```

Execution :

```
rps@rps-virtual-machine:~/signal_ex$ vim sig_defaultHandler.cpp
rps@rps-virtual-machine:~/signal_ex$ make sig_defaultHandler
g++     sig_defaultHandler.cpp   -o sig_defaultHandler
rps@rps-virtual-machine:~/signal_ex$ ./sig_defaultHandler
Press Ctrl+C ro generate SIGINT signal.
Waiting for signal.....
Ignoring SIGINT signal for the next 10 seconds.
^C^C^C^C^C^C^C
rps@rps-virtual-machine:~/signal_ex$
```