

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.

```

^C Interrupt signal (^C) received. Exiting program...
root@prps-virtual-machine: ~/signal$ cat signal.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void signalHandler(int signum) {
    cout << " Interrupt signal (" << signum <<") received. 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;
}
root@prps-virtual-machine: ~/signal$

```

```
rps@rps-virtual-machine: ~/signal.c$ vim signal.cpp  
rps@rps-virtual-machine:~/signal.c$ 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.c$ vim signal.cpp  
rps@rps-virtual-machine:~/signal.c$ make signal  
g++    signal.cpp -o signal  
rps@rps-virtual-machine:~/signal.c$ ./signal.cpp  
bash: ./signal.cpp: Permission denied  
rps@rps-virtual-machine:~/signal.c$ ./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.  
Program running .....Press Ctrl+C to exit.  
Program running .....Press Ctrl+C to exit.  
^C Interrupt signal(2) received. Exiting program...
```

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

```

#ps@ps-virtual-machine:~$ cat signal2.cpp
#include <iostream>
#include <signal>
#include <unistd.h>

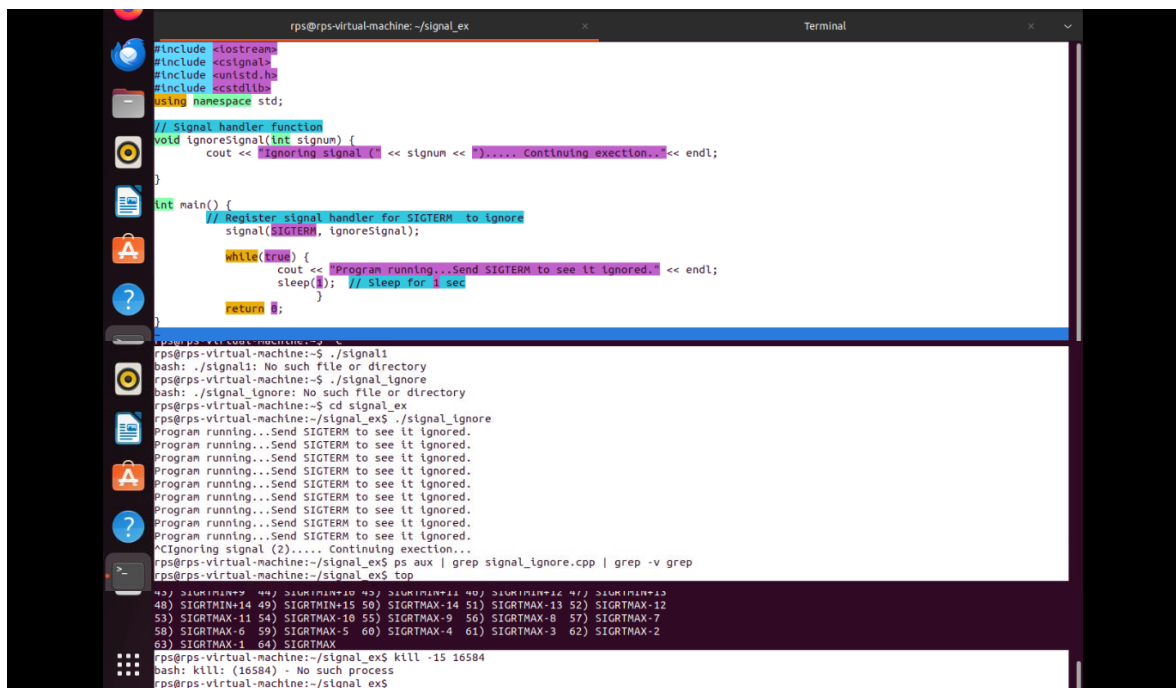
// Signal handler function
void signalHandler(int signum) {
    std::cout << "Whatever, Interrupt signal (" << signum << ") recieved.\n";
    // cleanup and close up stuff here
    // Terminate program
    exit(signum);
}

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(SIGSTP, Iamhere);
    while(true) {
        std::cout << "Running.....\n";
        sleep(1); // Sleep for 1 sec
    }
    return 0;
}

```


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



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

// Signal handler function
void ignoreSignal(int signum) {
    cout << "Ignoring signal (" << signum << ").... Continuing execution..." << endl;
}

int main() {
    // Register signal handler for SIGTERM to ignore
    signal(SIGTERM, ignoreSignal);

    while(true) {
        cout << "Program running...Send SIGTERM to see it ignored." << endl;
        sleep(1); // sleep for 1 sec
    }

    return 0;
}
```

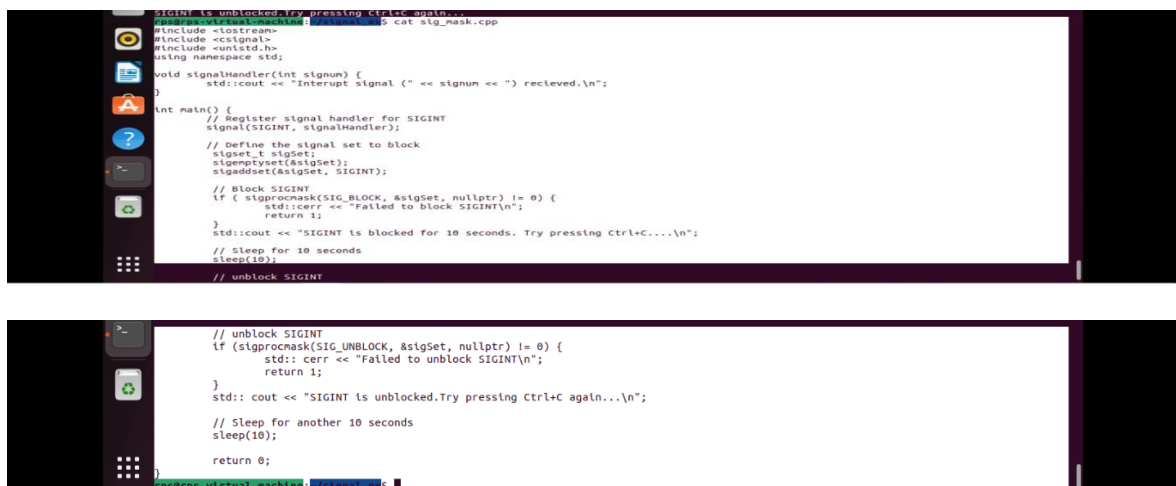
Terminal output:

```
rps@rps-virtual-machine:~/signal_ex$ g++ signal_ignore.cpp -o signal_ex
rps@rps-virtual-machine:~/signal_ex$ ./signal_ex
bash: ./signal: No such file or directory
rps@rps-virtual-machine:~/signal_ex$ ./signal_ignore
bash: ./signal_ignore: No such file or directory
rps@rps-virtual-machine:~/signal_ex$ cd signal_ex
rps@rps-virtual-machine:~/signal_ex$ ./signal_ignore
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
Program running...Send SIGTERM to see it ignored.
^CIgnoring signal (2).... Continuing execution...
rps@rps-virtual-machine:~/signal_ex$ ps aux | grep signal_ignore.cpp | grep -v grep
rps@rps-virtual-machine:~/signal_ex$ top
43j 3AUMIN+7 44j 3AUMIN+10 45j 3AUMIN+11 46j 3AUMIN+12 47j 3AUMIN+13
48) SIGRTMIN-14 49) SIGRTMIN-15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
rps@rps-virtual-machine:~/signal_ex$ kill -15 16584
bash: kill: (16584) - No such process
rps@rps-virtual-machine:~/signal_ex$
```

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.



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

void signalHandler(int signum) {
    std::cout << "Interrupt signal (" << signum << ") received.\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
    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;
}
```

Terminal output:

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

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

5. sig_mask1 (SIGTERM)

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

void signalHandler(int signum) {
    std::cout << "Interrupt 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";

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

rp@rp-virtual-machine:~/signal$
```

```
rp@rp-virtual-machine:~/signal$ ./sig_mask1
Interrupt signal (2) recieved.
SIGTERM is blocked during critical section.
Entering Critical section...
Exiting critical section...
SIGTERM is unblocked.
rp@rp-virtual-machine:~/signal$
```

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

```
rp@rp-virtual-machine:~/signal$ vim sig_mask1.cpp
rp@rp-virtual-machine:~/signal$ make sig_mask1
g++ sig_mask1.cpp -o sig_mask1
rp@rp-virtual-machine:~/signal$ ./sig_mask1
SIGINT is blocked for 10 seconds. Try pressing Ctrl+C....
SIGINT is unblocked. Try pressing Ctrl+C again....
rp@rp-virtual-machine:~/signal$
```

```
rps@rps-virtual-machine:~/signal$ vlm testsignalall.cpp
rps@rps-virtual-machine:~/signal$ make testsignalall
g++ testsignalall.cpp -o testsignalall
rps@rps-virtual-machine:~/signal$ vlm testsignalall.cpp
rps@rps-virtual-machine:~/signal$ make testsignalall
g++ testsignalall.cpp -o testsignalall
rps@rps-virtual-machine:~/signal$ ./testsignalall
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
^CReceived signal (2) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
^CReceived signal (3) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
^Z
[5]+  Stopped                  ./testsignalall
```

```
rps@rps-virtual-machine:~/signal$ kill -1 13923
rps@rps-virtual-machine:~/signal$ ps aux | grep testsignalall | grep -v grep
rps      13879  0.0  0.0  6056 3200 pts/1    T   12:55   0:00 ./testsignalall
rps      13896  0.0  0.0  6056 2944 pts/1    T   13:03   0:00 ./testsignalall
rps      13899  0.0  0.0  6056 3200 pts/1    T   13:04   0:00 ./testsignalall
rps      13922  0.0  0.0  6056 3200 pts/0    T   13:14   0:00 ./testsignalall
rps      13923  0.0  0.0  6056 3200 pts/1    S+  13:15   0:00 ./testsignalall
rps@rps-virtual-machine:~/signal$ kill -4 13923
rps@rps-virtual-machine:~/signal$ kill -5 13923
rps@rps-virtual-machine:~/signal$ kill -6 13923
rps@rps-virtual-machine:~/signal$ kill -7 13923
rps@rps-virtual-machine:~/signal$ kill -8 13923
rps@rps-virtual-machine:~/signal$ kill -9 13923
```

```
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (1) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (4) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (5) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (6) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (7) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Received signal (8) .....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Waiting for signal.....
Killed
rps@rps-virtual-machine:~/signal$ ./testsignalall
```

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 << "Interrupt signal (" << signum << ") received.\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$ ./sig_mask.cpp
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 << "Interrupt signal (" << signum << ") received.\n";
}

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

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

    // Block SIGTERM
    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";

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

    // unblock SIGTERM
    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$
```



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

```
rsps@rps-virtual-machine:~/signal_ex$ cat sig_mask2.cpp
#include <iostream>
#include <csignal>
#include <unistd.h>
using namespace std;

void error_signalHandler(int signum) {
    std::cerr << "Segmentation fault(" << signum << ") recieved.\n";
    exit(signum);
}

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

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

    std::cout << "Normal section starts\n";
    sleep(3);

    // Error handling routines (signals are blocked here)
    sigprocmask(SIG_BLOCK, &sigSet, nullptr);

    // Log error and clean up
    std::cerr << "Handling error.....\n";
    sleep(3);

    // unmask the signals
    sigprocmask(SIG_UNBLOCK, &sigSet, nullptr);

    int* p = nullptr;
    *p = 23;

    "sig_mask2.cpp" 38L, 780B
21,10-17 Top
```

```
rsps@rps-virtual-machine:~/signal_ex$ g++ sig_mask2.cpp
rsps@rps-virtual-machine:~/signal_ex$ make sig_mask2
g++ sig_mask2.cpp -o sig_mask2
rsps@rps-virtual-machine:~/signal_ex$ ./sig_mask2
Normal section starts
Handling error.....
Segmentation fault(11) recieved.
rsps@rps-virtual-machine:~/signal_ex$ ps aux | grep testsignalall | grep -v grep
rps      13879  0.0  0.0   6056  3200 pts/1    T   12:55   0:00 ./testsignalall
rps      13887  0.0  0.0   6056  3200 pts/1    T   12:58   0:00 ./testsignalall
rps      13896  0.0  0.0   6056  2944 pts/1    T   13:03   0:00 ./testsignalall
rps      13899  0.0  0.0   6056  3200 pts/1    T   13:04   0:00 ./testsignalall
rps      13922  0.0  0.0   6056  3200 pts/0    T   13:14   0:00 ./testsignalall
```

3. Files Manipulation using signal Handlers

Execution :

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

File tempfile.txt created.
Press Ctrl+C to register the signal handler....
^C
Interrupt signal (2) recieved.
File tempfile.txt deleted successfully.
rsps@rps-virtual-machine:~/signal_ex$
```

```

#include <iostream>
#include <signal>
#include <cstdlib>
#include <unistd.h>
#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";

// Infinite 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
#include <iostream>
#include <signal>
#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 to 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 $ vlm 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 to generate SIGINT signal.
Waiting for signal....
Ignoring SIGINT signal for the next 10 seconds.
rps@rps-virtual-machine: ~/signal_ex $

```