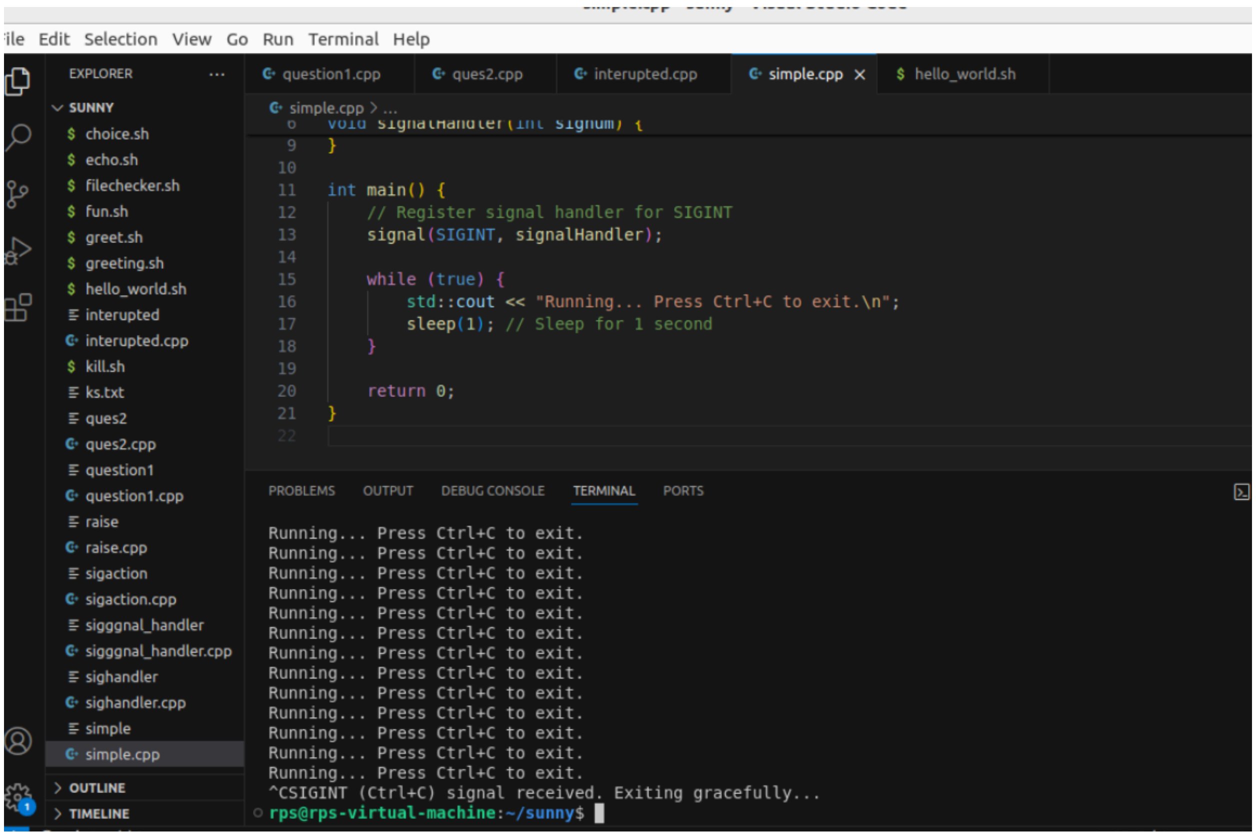
Q.Simple Signal Handler: Write a C++ program that handles the SIGINT signal (Ctrl+C) gracefully by printing a custom message before exiting.



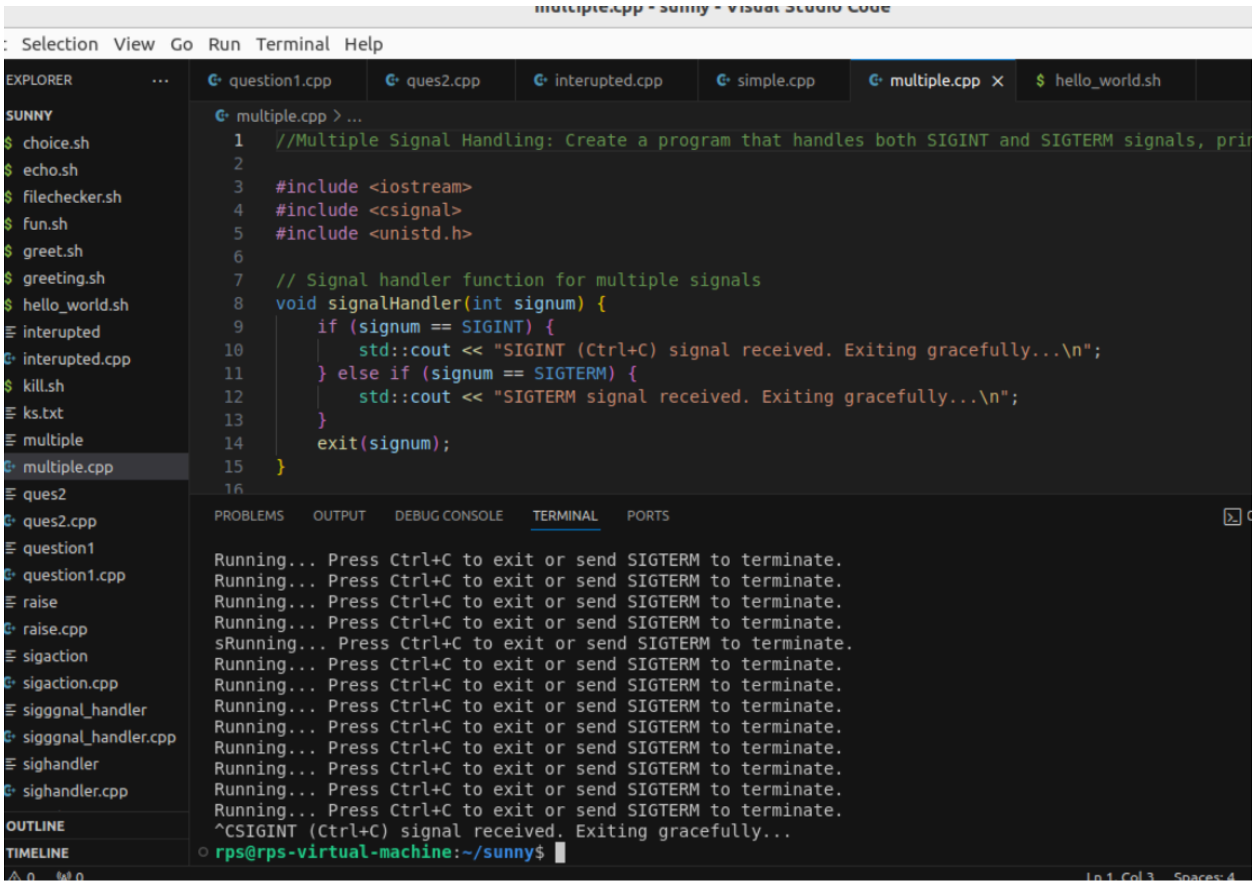
```
void signalHandler(int signum) {
}

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

    while (true) {
        std::cout << "Running... Press Ctrl+C to exit.\n";
        sleep(1); // Sleep for 1 second
    }

    return 0;
}
```

```
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
Running... Press Ctrl+C to exit.
^CSIGINT (Ctrl+C) signal received. Exiting gracefully...
rps@rps-virtual-machine:~/sunny$
```

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

```
#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function for multiple signals
void signalHandler(int signum) {
   if (signum == SIGINT) {
      std::cout << "SIGINT (Ctrl+C) signal received. Exiting gracefully...\n";
   } else if (signum == SIGTERM) {
      std::cout << "SIGTERM signal received. Exiting gracefully...\n";
   }
   exit(signum);
}
```

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

    while (true) {
        std::cout << "Running... Press Ctrl+C to exit or send SIGTERM to terminate.\n";
        sleep(1); // Sleep for 1 second
    }

    return 0;
}
```



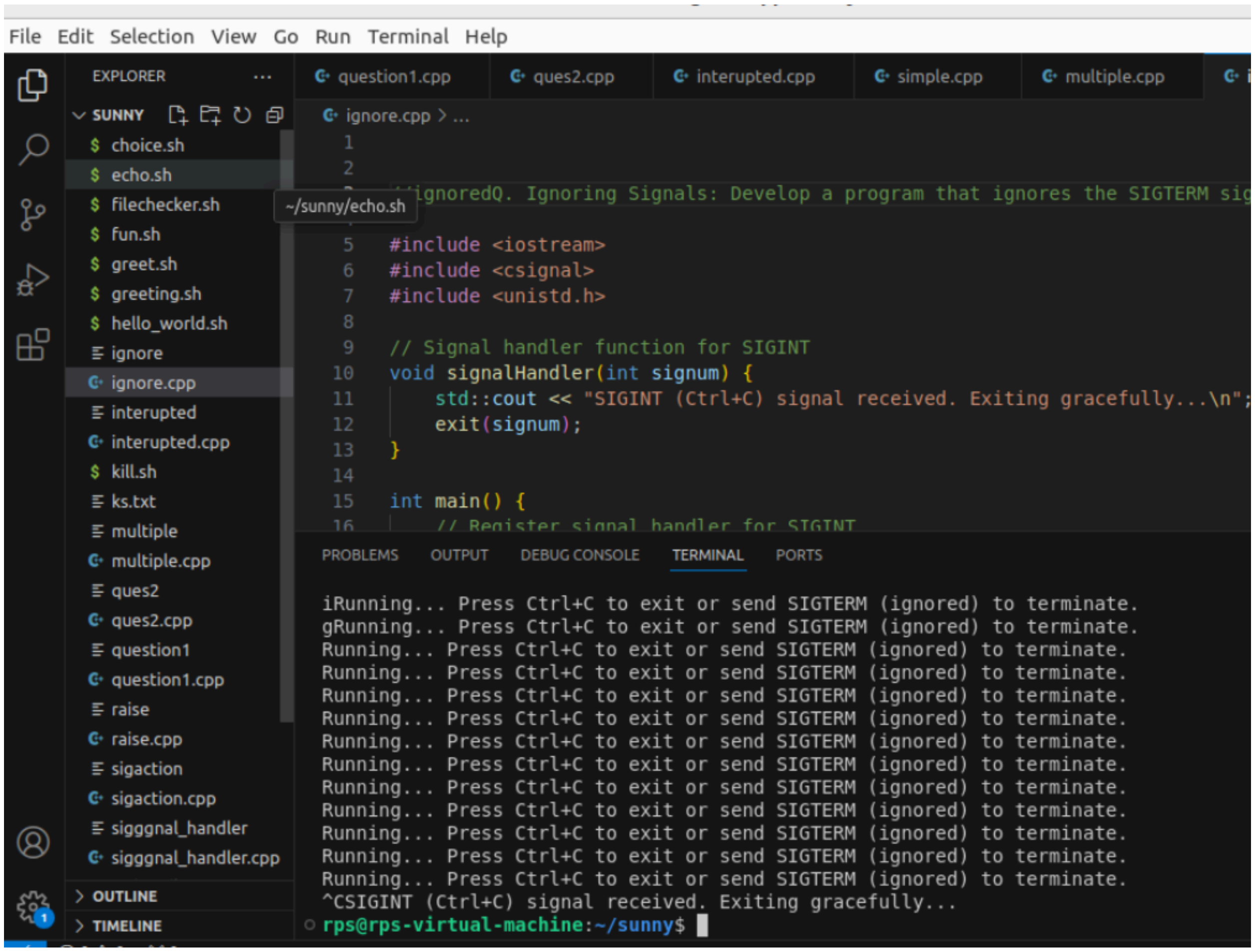
```
//Multiple Signal Handling: Create a program that handles both SIGINT and SIGTERM signals, pri

#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function for multiple signals
void signalHandler(int signum) {
    if (signum == SIGINT) {
        std::cout << "SIGINT (Ctrl+C) signal received. Exiting gracefully...\n";
    } else if (signum == SIGTERM) {
        std::cout << "SIGTERM signal received. Exiting gracefully...\n";
    }
    exit(signum);
}
```

```
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
sRunning... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
^CSIGINT (Ctrl+C) signal received. Exiting gracefully...
rps@rps-virtual-machine:~/sunny$
```

Q. 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>
```

```
// Signal handler function for SIGINT
void signalHandler(int signum) {
   std::cout << "SIGINT (Ctrl+C) signal received. Exiting gracefully...\n";
   exit(signum);
}

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

   // Ignore SIGTERM signal
   signal(SIGTERM, SIG_IGN);

   while (true) {
      std::cout << "Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.\n";
      sleep(1); // Sleep for 1 second
   }

   return 0;
}
```

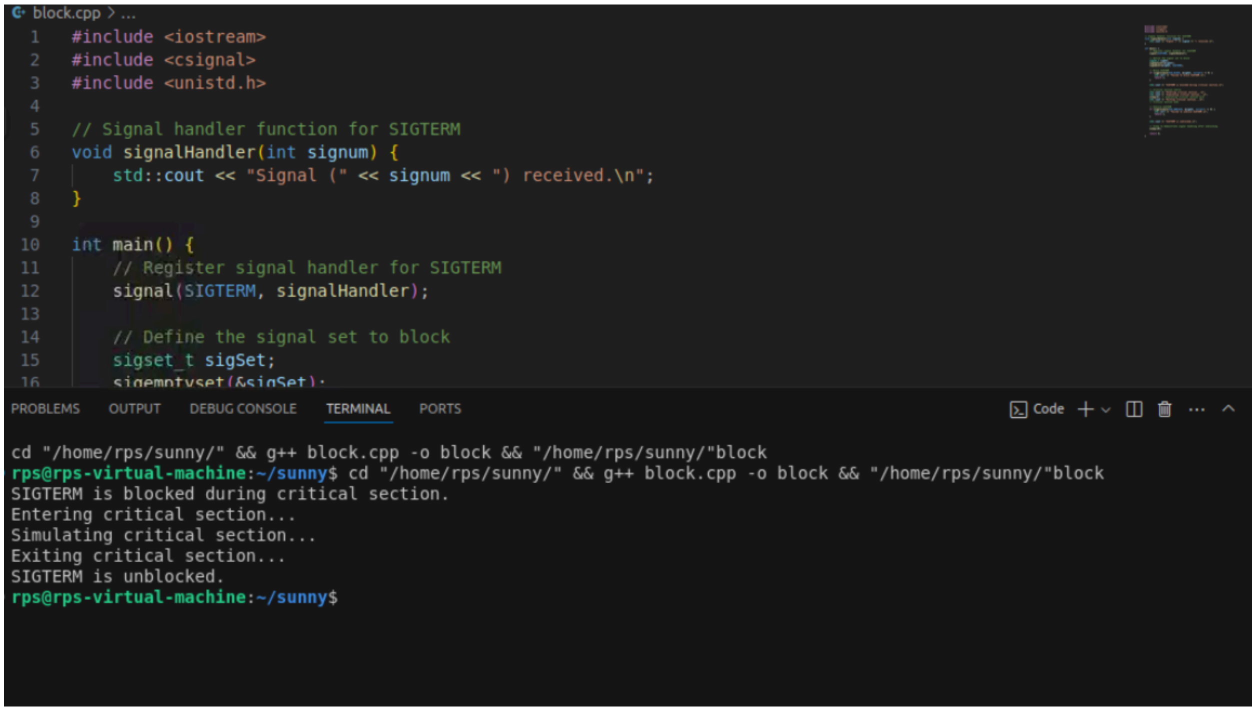


```
//ignoredQ. Ignoring Signals: Develop a program that ignores the SIGTERM sig

#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function for SIGINT
void signalHandler(int signum) {
    std::cout << "SIGINT (Ctrl+C) signal received. Exiting gracefully...\n";
    exit(signum);
}

int main() {
    // Register signal handler for SIGINT
```

```
iRunning... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
gRunning... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
Running... Press Ctrl+C to exit or send SIGTERM (ignored) to terminate.
^CSIGINT (Ctrl+C) signal received. Exiting gracefully...
rps@rps-virtual-machine:~/sunny$
```

```
#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function for SIGTERM
void signalHandler(int signum) {
   std::cout << "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";
    std::cout << "Simulating critical section...\n";
    sleep(5); // Simulating critical section work
    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;
}
```

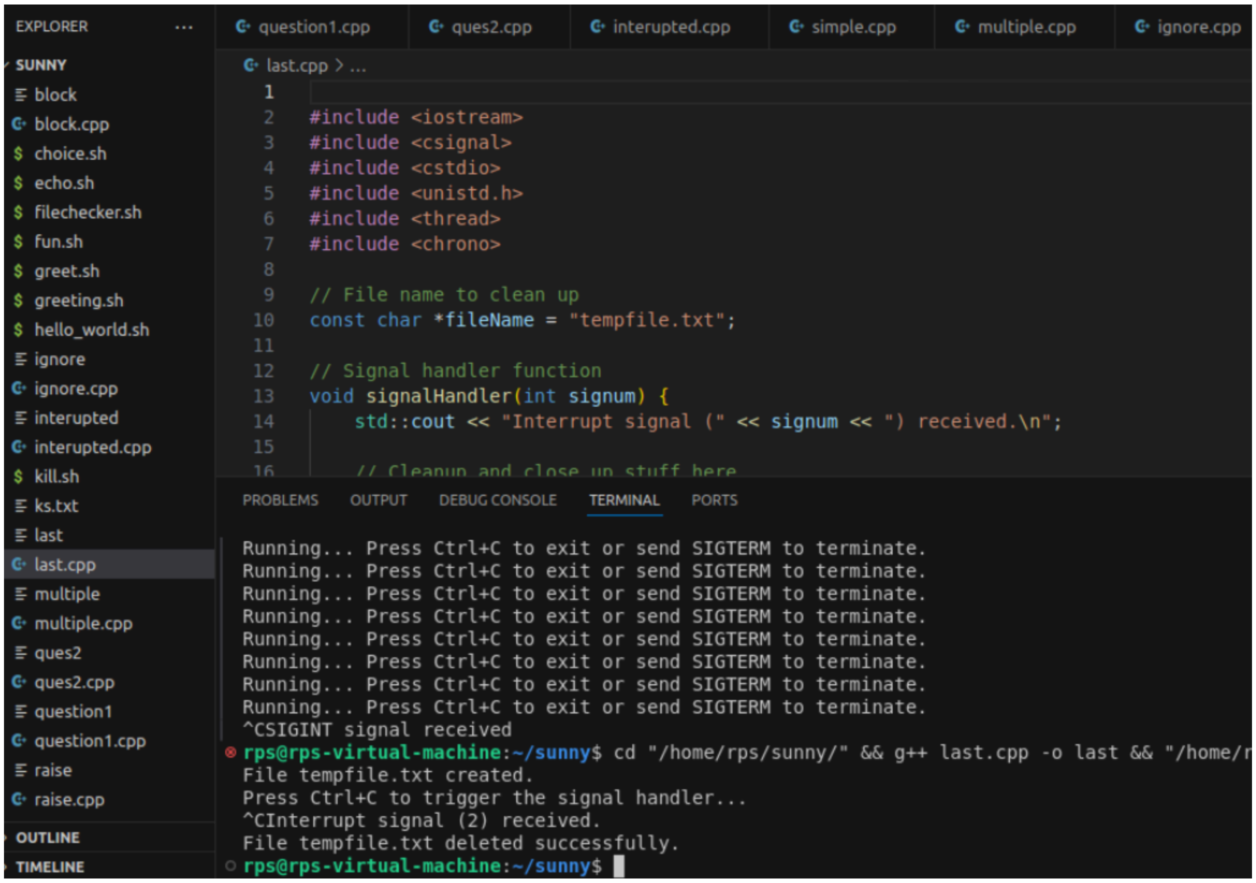

block.cpp > ...

```cpp
#include <iostream>
#include <csignal>
#include <unistd.h>

// Signal handler function for SIGTERM
void signalHandler(int signum) {
    std::cout << "Signal (" << signum << ") received.\n";
}

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

    // Define the signal set to block
    sigset_t sigSet;
```

```
cd "/home/rps/sunny/" && g++ block.cpp -o block && "/home/rps/sunny/"block
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ block.cpp -o block && "/home/rps/sunny/"block
SIGTERM is blocked during critical section.
Entering critical section...
Simulating critical section...
Exiting critical section...
SIGTERM is unblocked.
rps@rps-virtual-machine:~/sunny$
```

```cpp
#include <iostream>
#include <csignal>
#include <cstdio>
#include <unistd.h>
#include <thread>
#include <chrono>

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

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

    // Cleanup and close 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 removed
    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";

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

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

    // Infinite loop to keep the program running
    while (true) {
        // Sleep to prevent high CPU usage
        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    return 0;
}
```



```cpp

#include <iostream>
#include <csignal>
#include <cstdio>
#include <unistd.h>
#include <thread>
#include <chrono>

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

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

    // Cleanup and close up stuff here
```

```
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
Running... Press Ctrl+C to exit or send SIGTERM to terminate.
^CSIGINT signal received
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ last.cpp -o last && "/home/r
File tempfile.txt created.
Press Ctrl+C to trigger the signal handler...
^CInterrupt signal (2) received.
File tempfile.txt deleted successfully.
rps@rps-virtual-machine:~/sunny$
```