task is to implement a function to process a signal and mark the processed elements using a specific marker. The signal is represented as a vector of integers. You need to:

Define a marker value (SIGPROCMARK) to mark the processed signal elements.

Implement a function processSignal that processes each element of the signal by doubling its value and then marking it with SIGPROCMARK.

Implement a function displaySignal to print the signal values to the console.

Demonstrate the usage of these functions in a main function with an example signal.

Requirements:

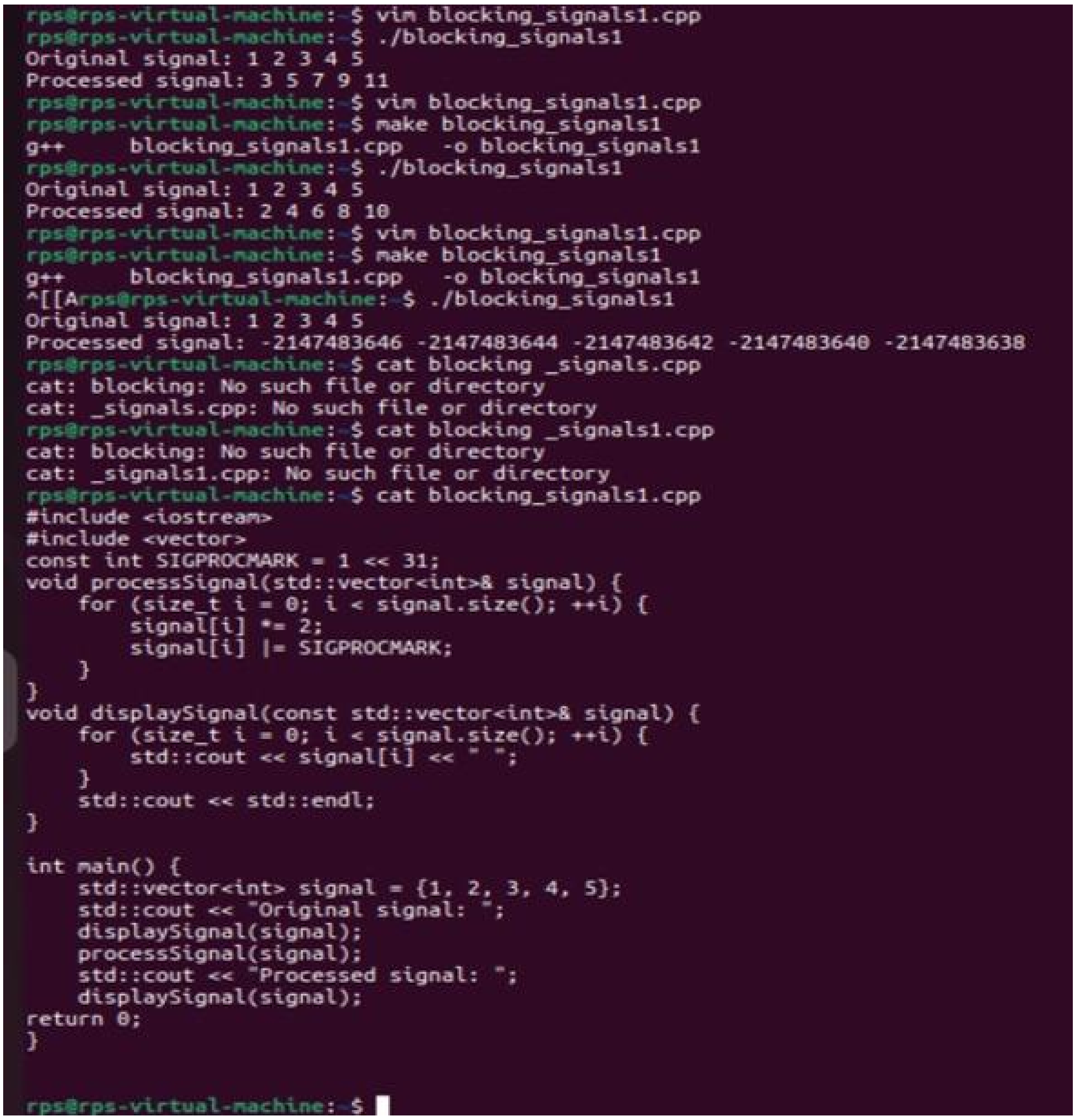
The marker value should be defined as a constant.

The processSignal function should use bitwise operations to mark the processed elements.

The displaySignal function should print the signal values separated by spaces.

Input:

An example signal represented as a vector of integers, e.g., {1, 2, 3, 4, 5}.



```
rps@rps-virtual-machine:~$ vim blocking_signals1.cpp
rps@rps-virtual-machine:~$ ./blocking_signals1
Original signal: 1 2 3 4 5
Processed signal: 3 5 7 9 11
rps@rps-virtual-machine:~$ vim blocking_signals1.cpp
rps@rps-virtual-machine:~$ make blocking_signals1
g++     blocking_signals1.cpp    -o blocking_signals1
rps@rps-virtual-machine:~$ ./blocking_signals1
Original signal: 1 2 3 4 5
Processed signal: 2 4 6 8 10
rps@rps-virtual-machine:~$ vim blocking_signals1.cpp
rps@rps-virtual-machine:~$ make blocking_signals1
g++     blocking_signals1.cpp    -o blocking_signals1
^[[Arps@rps-virtual-machine:~$ ./blocking_signals1
Original signal: 1 2 3 4 5
Processed signal: -2147483646 -2147483644 -2147483642 -2147483640 -2147483638
rps@rps-virtual-machine:~$ cat blocking _signals.cpp
cat: blocking: No such file or directory
cat: _signals.cpp: No such file or directory
rps@rps-virtual-machine:~$ cat blocking _signals1.cpp
cat: blocking: No such file or directory
cat: _signals1.cpp: No such file or directory
rps@rps-virtual-machine:~$ cat blocking_signals1.cpp
#include <iostream>
#include <vector>
const int SIGPROCMARK = 1 << 31;
void processSignal(std::vector<int>& signal) {
    for (size_t i = 0; i < signal.size(); ++i) {
        signal[i] *= 2;
        signal[i] |= SIGPROCMARK;
    }
}
void displaySignal(const std::vector<int>& signal) {
    for (size_t i = 0; i < signal.size(); ++i) {
        std::cout << signal[i] << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> signal = {1, 2, 3, 4, 5};
    std::cout << "Original signal: ";
    displaySignal(signal);
    processSignal(signal);
    std::cout << "Processed signal: ";
    displaySignal(signal);
return 0;
}


rps@rps-virtual-machine:~$ 
```

Signal Processing with Threshold Marking

You are tasked with extending the signal processing project to include a threshold marking mechanism. Your goal is to:

Define a marker value (SIGPROCMARK) to mark the processed signal elements.

Implement a function processSignalWithThreshold that processes each element of the signal by doubling its value only if it is greater than a given threshold, and then marking it with SIGPROCMARK.

Implement a function displaySignal to print the signal values to the console.

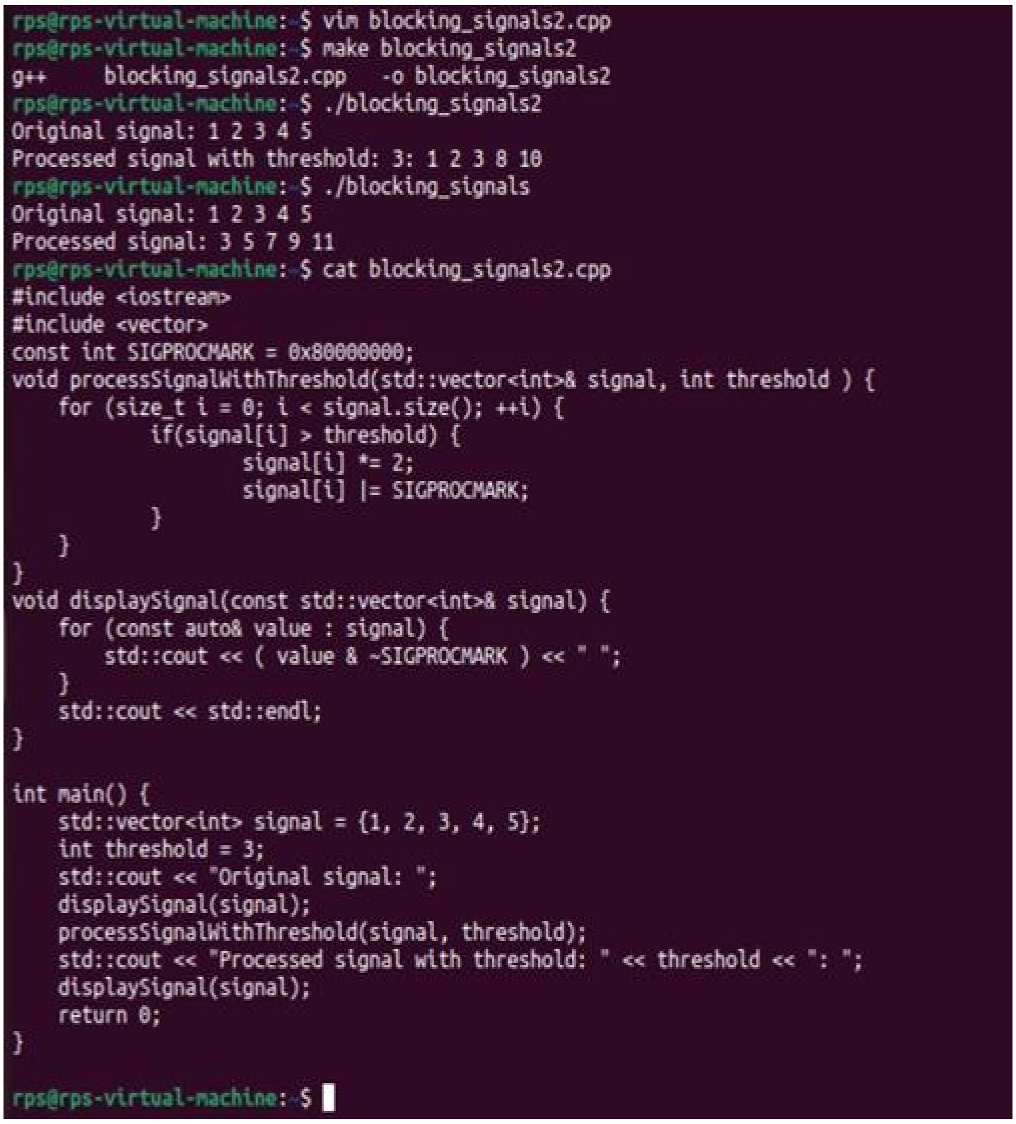
Demonstrate the usage of these functions in a main function with an example signal and a threshold value.

Requirements:

The marker value should be defined as a constant.

The processSignalWithThreshold function should double the value of each element that exceeds the threshold and use bitwise operations to mark the processed elements.

The displaySignal function should print the signal values separated by spaces.

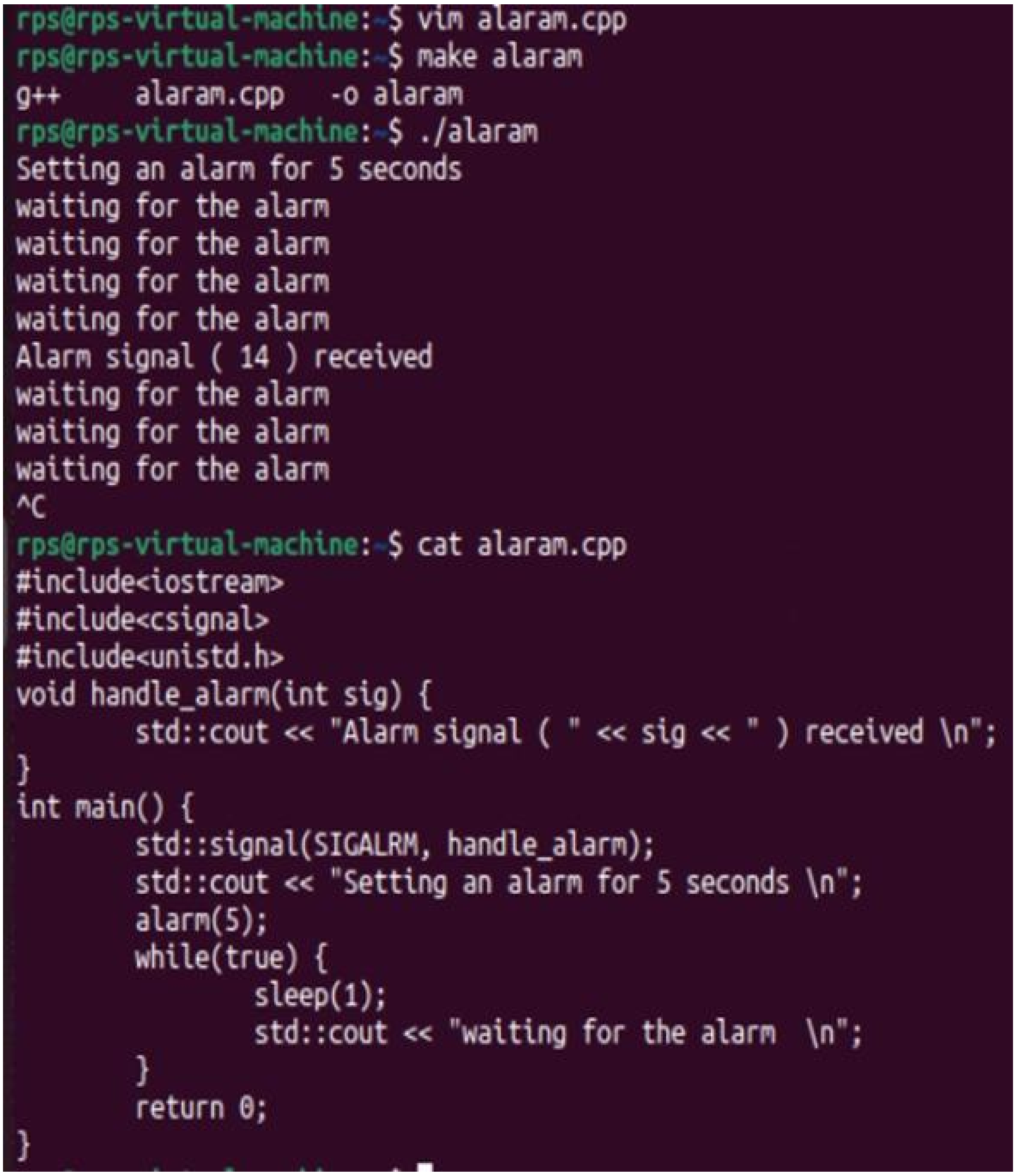


```
rps@rps-virtual-machine:~$ vim blocking_signals2.cpp
rps@rps-virtual-machine:~$ make blocking_signals2
g++     blocking_signals2.cpp    -o blocking_signals2
rps@rps-virtual-machine:~$ ./blocking_signals2
Original signal: 1 2 3 4 5
Processed signal with threshold: 3: 1 2 3 8 10
rps@rps-virtual-machine:~$ ./blocking_signals
Original signal: 1 2 3 4 5
Processed signal: 3 5 7 9 11
rps@rps-virtual-machine:~$ cat blocking_signals2.cpp
#include <iostream>
#include <vector>
const int SIGPROCMARK = 0x80000000;
void processSignalWithThreshold(std::vector<int>& signal, int threshold ) {
    for (size_t i = 0; i < signal.size(); ++i) {
            if(signal[i] > threshold) {
                    signal[i] *= 2;
                    signal[i] |= SIGPROCMARK;
            }
    }
}
void displaySignal(const std::vector<int>& signal) {
    for (const auto& value : signal) {
        std::cout << ( value & ~SIGPROCMARK ) << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::vector<int> signal = {1, 2, 3, 4, 5};
    int threshold = 3;
    std::cout << "Original signal: ";
    displaySignal(signal);
    processSignalWithThreshold(signal, threshold);
    std::cout << "Processed signal with threshold: " << threshold << ": ";
    displaySignal(signal);
    return 0;
}

rps@rps-virtual-machine:~$ 
```

Alarm Program:



```
rps@rps-virtual-machine:~$ vim alaram.cpp
rps@rps-virtual-machine:~$ make alaram
g++     alaram.cpp   -o alaram
rps@rps-virtual-machine:~$ ./alaram
Setting an alarm for 5 seconds
waiting for the alarm
waiting for the alarm
waiting for the alarm
waiting for the alarm
Alarm signal ( 14 ) received
waiting for the alarm
waiting for the alarm
waiting for the alarm
^C
rps@rps-virtual-machine:~$ cat alaram.cpp
#include<iostream>
#include<csignal>
#include<unistd.h>
void handle_alarm(int sig) {
        std::cout << "Alarm signal ( " << sig << " ) received \n";
}
int main() {
        std::signal(SIGALRM, handle_alarm);
        std::cout << "Setting an alarm for 5 seconds \n";
        alarm(5);
        while(true) {
                sleep(1);
                std::cout << "waiting for the alarm  \n";
        }
        return 0;
}
```

Develop a C++ application that demonstrates effective signal handling using SIGALRM, SIGDEFAULT, and SIG_IGN. The program should:

Set up a timer using alarm() to generate a SIGALRM signal after a specified interval.

Define a signal handler function to process the SIGALRM signal and perform specific actions, such as printing a message, updating a counter, or triggering an event.

Implement logic to handle other signals (e.g., SIGINT, SIGTERM) using SIGDEFAULT or SIG_IGN as appropriate.

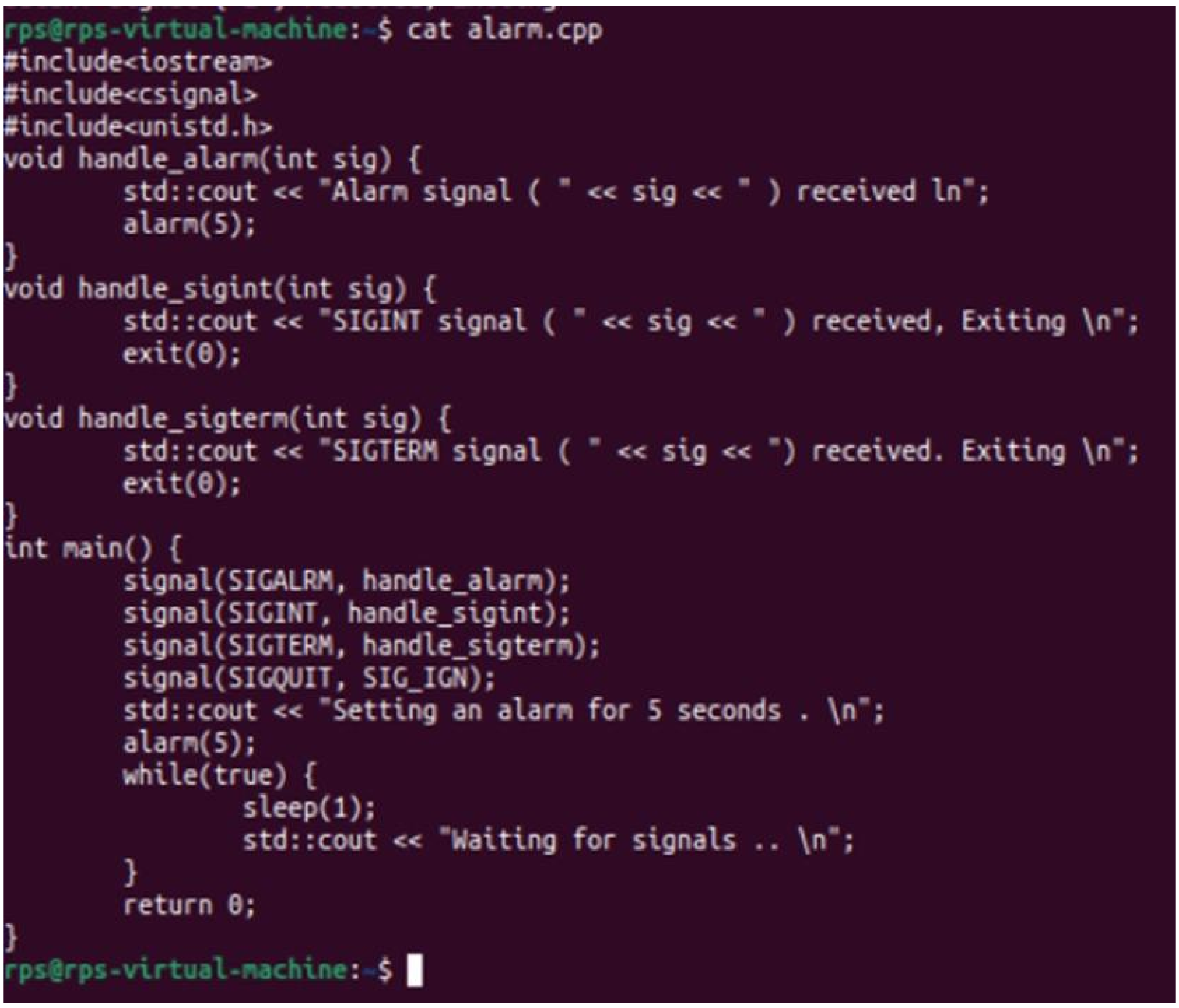
Explore the behavior of the application under different signal combinations and handling strategies.

Additional Considerations:

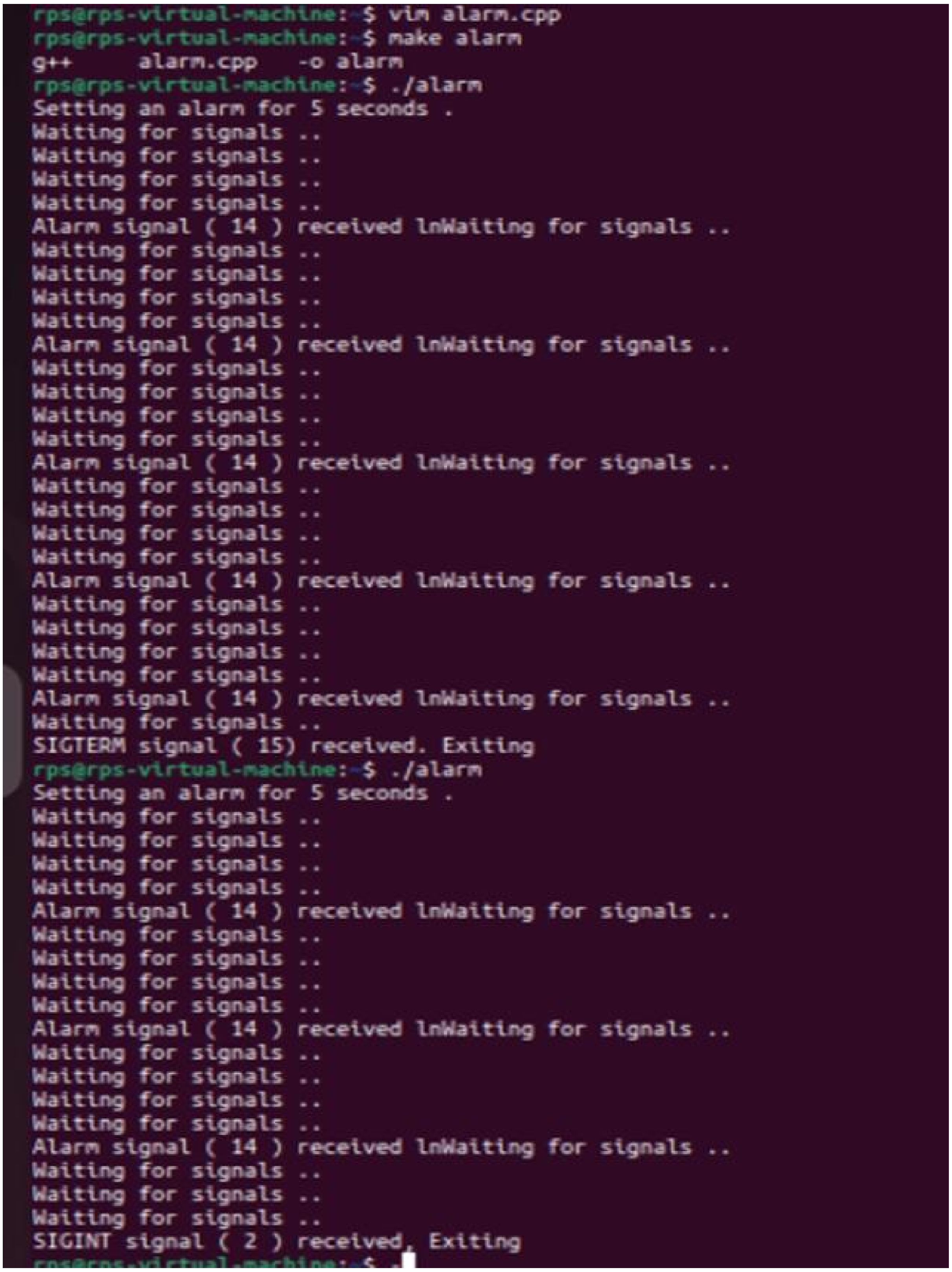
Consider the impact of signal handling on program execution and potential race conditions.

Investigate the use of sigaction for more advanced signal handling capabilities.

Explore the application of signal handling in real-world scenarios, such as timeouts, asynchronous events, and error handling.

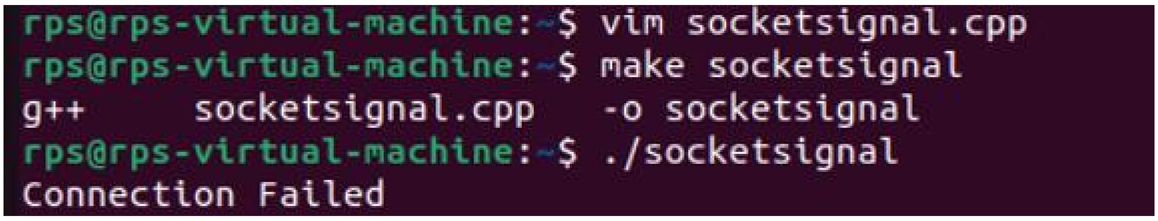


```
rps@rps-virtual-machine:~$ cat alarm.cpp
#include<iostream>
#include<csignal>
#include<unistd.h>
void handle_alarm(int sig) {
        std::cout << "Alarm signal ( " << sig << " ) received ln";
        alarm(5);
}
void handle_sigint(int sig) {
        std::cout << "SIGINT signal ( " << sig << " ) received, Exiting \n";
        exit(0);
}
void handle_sigterm(int sig) {
        std::cout << "SIGTERM signal ( " << sig << ") received. Exiting \n";
        exit(0);
}
int main() {
        signal(SIGALRM, handle_alarm);
        signal(SIGINT, handle_sigint);
        signal(SIGTERM, handle_sigterm);
        signal(SIGQUIT, SIG_IGN);
        std::cout << "Setting an alarm for 5 seconds . \n";
        alarm(5);
        while(true) {
                sleep(1);
                std::cout << "Waiting for signals .. \n";
        }
        return 0;
}
rps@rps-virtual-machine:~$
```

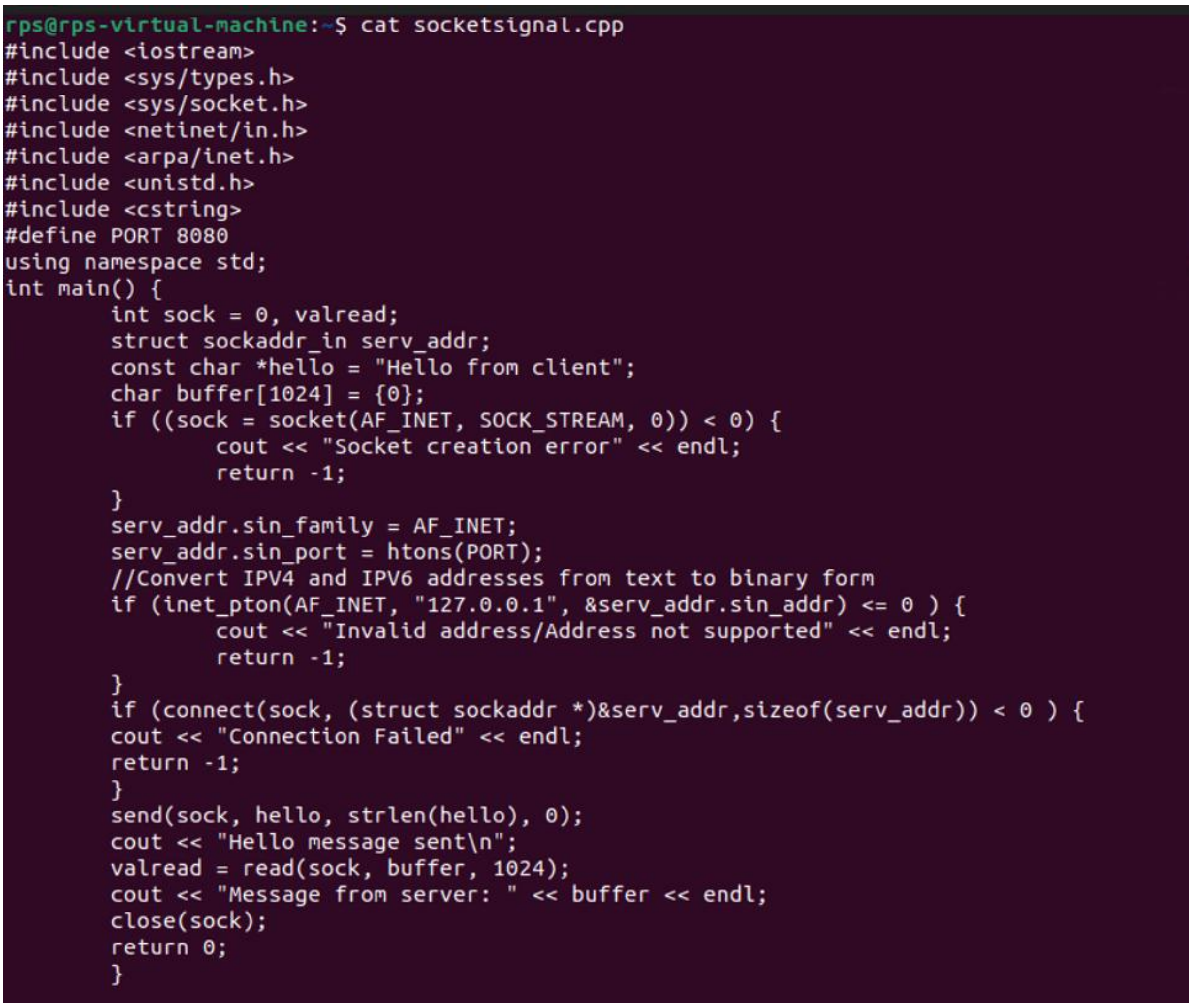


```
rps@rps-virtual-machine:~$ vim alarm.cpp
rps@rps-virtual-machine:~$ make alarm
g++     alarm.cpp   -o alarm
rps@rps-virtual-machine:~$ ./alarm
Setting an alarm for 5 seconds .
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
SIGTERM signal ( 15) received. Exiting
rps@rps-virtual-machine:~$ ./alarm
Setting an alarm for 5 seconds .
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
Alarm signal ( 14 ) received lnWaiting for signals ..
Waiting for signals ..
Waiting for signals ..
Waiting for signals ..
SIGINT signal ( 2 ) received. Exiting
rps@rps-virtual-machine:~$
```

Socketsignal:

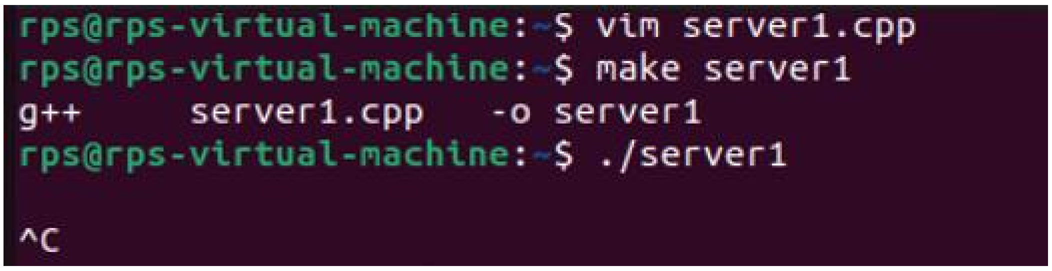


```
rps@rps-virtual-machine:~$ vim socketsignal.cpp
rps@rps-virtual-machine:~$ make socketsignal
g++     socketsignal.cpp   -o socketsignal
rps@rps-virtual-machine:~$ ./socketsignal
Connection Failed
```



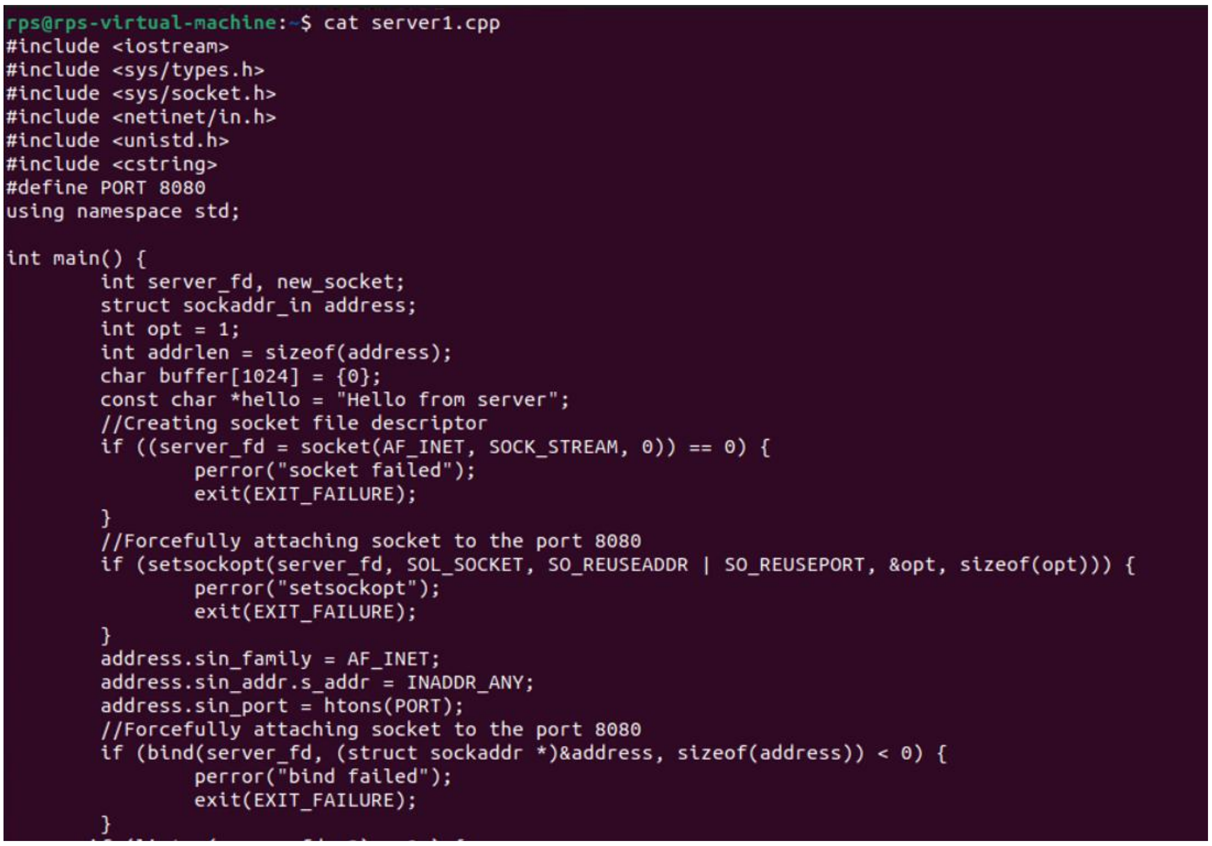
```
rps@rps-virtual-machine:~$ cat socketsignal.cpp
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#define PORT 8080
using namespace std;
int main() {
        int sock = 0, valread;
        struct sockaddr_in serv_addr;
        const char *hello = "Hello from client";
        char buffer[1024] = {0};
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
                cout << "Socket creation error" << endl;
                return -1;
        }
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);
        //Convert IPV4 and IPV6 addresses from text to binary form
        if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0 ) {
                cout << "Invalid address/Address not supported" << endl;
                return -1;
        }
        if (connect(sock, (struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0 ) {
        cout << "Connection Failed" << endl;
        return -1;
        }
        send(sock, hello, strlen(hello), 0);
        cout << "Hello message sent\n";
        valread = read(sock, buffer, 1024);
        cout << "Message from server: " << buffer << endl;
        close(sock);
        return 0;
        }
```

Server:



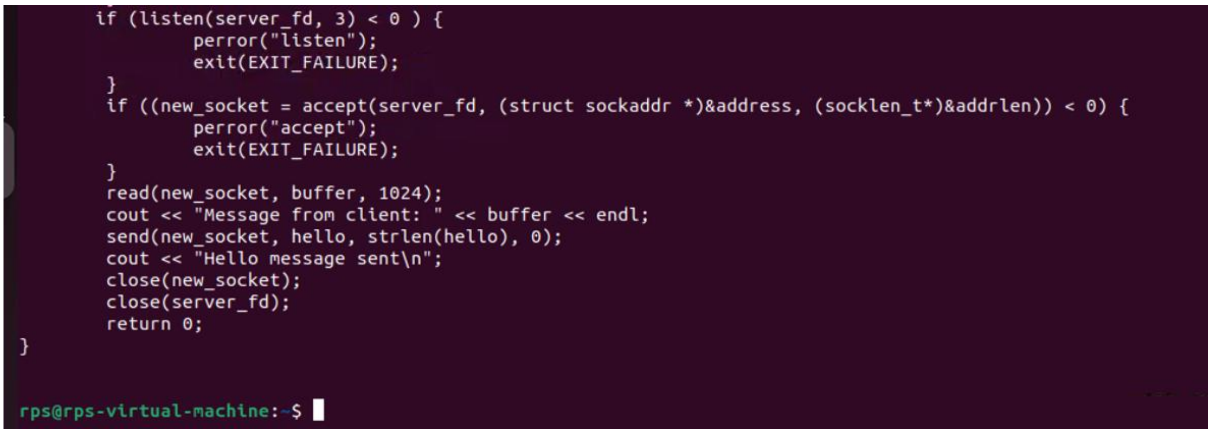
```
rps@rps-virtual-machine:~$ vim server1.cpp
rps@rps-virtual-machine:~$ make server1
g++     server1.cpp    -o server1
rps@rps-virtual-machine:~$ ./server1

^C
```



```
rps@rps-virtual-machine:~$ cat server1.cpp
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>
#define PORT 8080
using namespace std;

int main() {
        int server_fd, new_socket;
        struct sockaddr_in address;
        int opt = 1;
        int addrlen = sizeof(address);
        char buffer[1024] = {0};
        const char *hello = "Hello from server";
        //Creating socket file descriptor
        if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
                perror("socket failed");
                exit(EXIT_FAILURE);
        }
        //Forcefully attaching socket to the port 8080
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
                perror("setsockopt");
                exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons(PORT);
        //Forcefully attaching socket to the port 8080
        if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
                perror("bind failed");
                exit(EXIT_FAILURE);
        }
```



```
        if (listen(server_fd, 3) < 0 ) {
                perror("listen");
                exit(EXIT_FAILURE);
        }
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
                perror("accept");
                exit(EXIT_FAILURE);
        }
        read(new_socket, buffer, 1024);
        cout << "Message from client: " << buffer << endl;
        send(new_socket, hello, strlen(hello), 0);
        cout << "Hello message sent\n";
        close(new_socket);
        close(server_fd);
        return 0;
}

rps@rps-virtual-machine:~$ 
```