

SUNNY KUMAR  
24 JULY TASK  
LSP

### **process a signal and mark**

```
#include <iostream>
#include <vector>

// Define a marker value for processed signals
const int SIGPROCMARK = 1;

// Function to process a signal and mark it
void processSignal(std::vector<int>& signal) {
    for (size_t i = 0; i < signal.size(); ++i) {
        // Process the signal (example: double the value)
        signal[i] *= 2;

        // Mark the processed signal
        signal[i] |= SIGPROCMARK;
    }
}

// Function to display the signal
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() {
    // Example signal
    std::vector<int> signal = {1, 2, 3, 4, 5};

    std::cout << "Original signal: ";
    displaySignal(signal);

    // Process the signal
    processSignal(signal);

    std::cout << "Processed signal: ";
```

```

displaySignal(signal);

return 0;
}

```

The screenshot shows a C++ IDE with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer lists various files, including 'processsignal.cpp'. The code editor shows the implementation of the 'processSignal' and 'displaySignal' functions. The terminal shows the command to compile and run the program, followed by the output of the program.

```

// processsignal.cpp
26 int main() {
27     std::vector<int> signal = {1, 2, 3, 4, 5};
28
29     std::cout << "Original signal: ";
30     displaySignal(signal);
31
32     // Process the signal
33     processSignal(signal);
34
35     std::cout << "Processed signal: ";
36     displaySignal(signal);
37
38     return 0;
39 }
40
41

```

```

cd "/home/rps/sunny/" && g++ processsignal.cpp -o processsignal && "/home/rps/sunny/"processsignal
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ processsignal.cpp -o processsignal && "/home/rps/sunny/"
esssignal
Original signal: 1 2 3 4 5
Processed signal: 3 5 7 9 11
rps@rps-virtual-machine:~/sunny$

```

Q. 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}.

```

#include <iostream>
#include <vector>

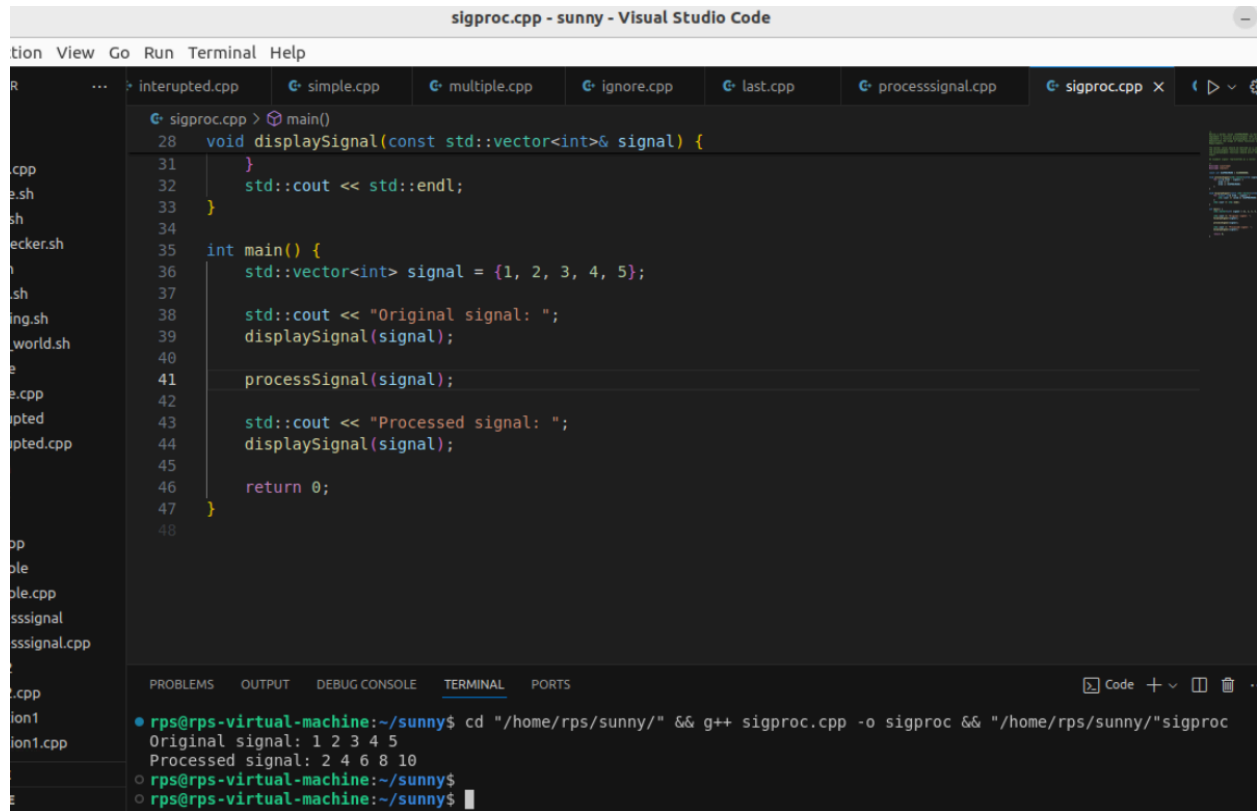
```

```
const int SIGPROCMARK = 0x80000000;
```

```
void processSignal(std::vector<int>& signal) {  
    for (int& elem : signal) {  
  
        elem *= 2;  
  
        elem |= SIGPROCMARK;  
    }  
}
```

```
void displaySignal(const std::vector<int>& signal) {  
    for (const int& elem : signal) {  
  
        std::cout << (elem & ~SIGPROCMARK) << " ";  
    }  
    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;  
}
```



The screenshot shows the Visual Studio Code editor with the file `sigproc.cpp` open. The code defines a `displaySignal` function and a `main` function. The `main` function creates a vector `signal = {1, 2, 3, 4, 5}`, prints it, calls `processSignal`, prints the result, and returns 0. The terminal at the bottom shows the command to compile and run the program, and the output of the program.

```
sigproc.cpp - sunny - Visual Studio Code
tion View Go Run Terminal Help
R ... > interrupted.cpp < simple.cpp < multiple.cpp < ignore.cpp < last.cpp < processsignal.cpp < sigproc.cpp x < > {
sigproc.cpp > main()
28 void displaySignal(const std::vector<int>& signal) {
31 }
32 std::cout << std::endl;
33 }
34
35 int main() {
36 std::vector<int> signal = {1, 2, 3, 4, 5};
37
38 std::cout << "Original signal: ";
39 displaySignal(signal);
40
41 processSignal(signal);
42
43 std::cout << "Processed signal: ";
44 displaySignal(signal);
45
46 return 0;
47 }
48

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ sigproc.cpp -o sigproc && "/home/rps/sunny/"sigproc
Original signal: 1 2 3 4 5
Processed signal: 2 4 6 8 10
rps@rps-virtual-machine:~/sunny$
rps@rps-virtual-machine:~/sunny$
```

## Q. 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.

```
#include <iostream>
```

```
#include <vector>
```

```

const int SIGPROCMARK = 0x80000000;

void processSignalWithThreshold(std::vector<int>& signal, int threshold) {
    for (int& elem : signal) {
        if (elem > threshold) {

            elem *= 2;

            elem |= SIGPROCMARK;
        }
    }
}

void displaySignal(const std::vector<int>& signal) {
    for (const int& elem : signal) {

        std::cout << (elem & ~SIGPROCMARK) << " ";
    }
    std::cout << std::endl;
}

int main() {

    std::vector<int> signal = {1, 2, 3, 4, 5};

    int threshold = 3;

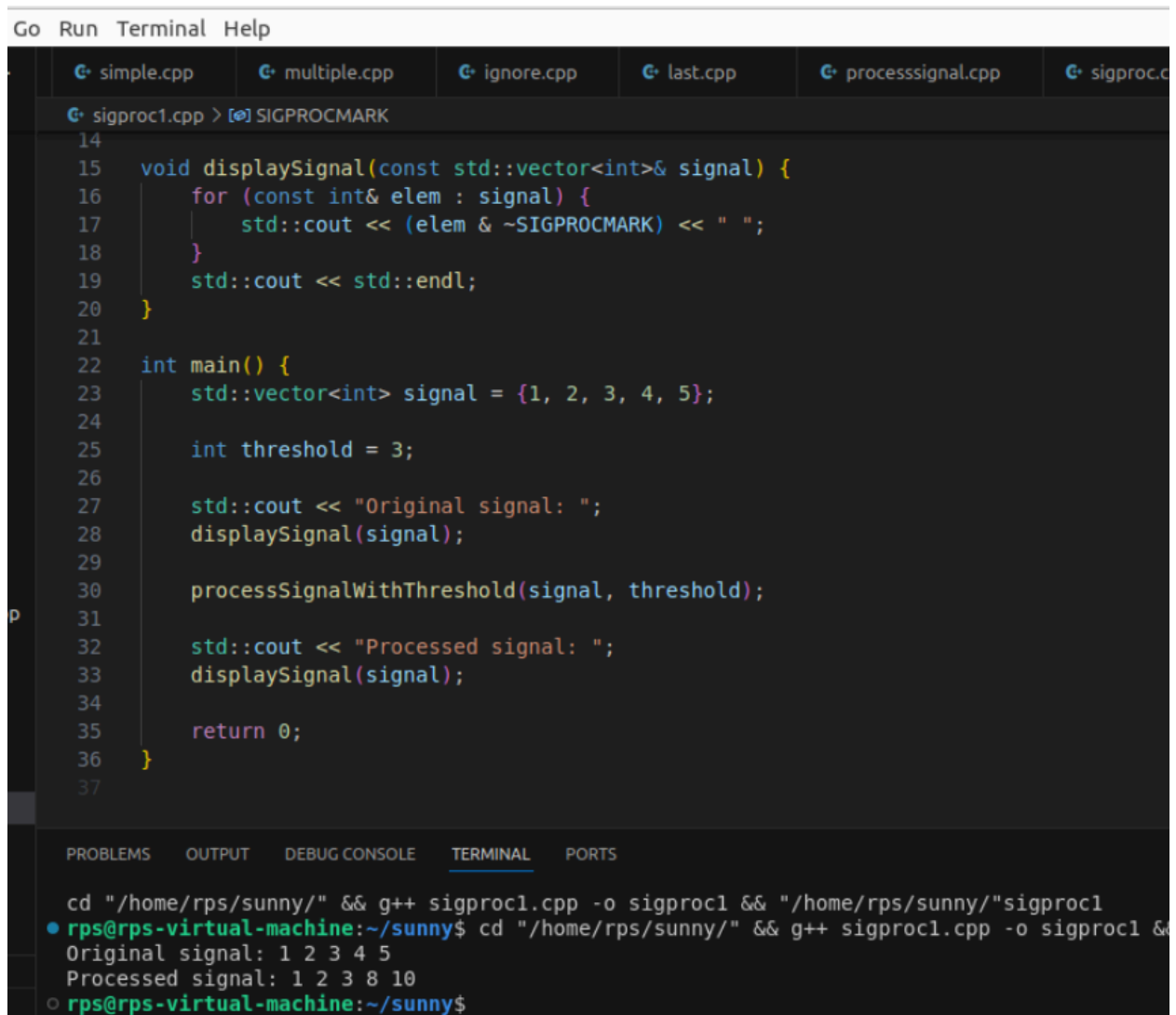
    std::cout << "Original signal: ";
    displaySignal(signal);

    processSignalWithThreshold(signal, threshold);

    std::cout << "Processed signal: ";
    displaySignal(signal);

    return 0;
}

```



```
Go Run Terminal Help
simple.cpp multiple.cpp ignore.cpp last.cpp processsignal.cpp sigproc.c
sigproc1.cpp > [SIGPROC] MARK
14
15 void displaySignal(const std::vector<int>& signal) {
16     for (const int& elem : signal) {
17         std::cout << (elem & ~SIGPROC) << " ";
18     }
19     std::cout << std::endl;
20 }
21
22 int main() {
23     std::vector<int> signal = {1, 2, 3, 4, 5};
24
25     int threshold = 3;
26
27     std::cout << "Original signal: ";
28     displaySignal(signal);
29
30     processSignalWithThreshold(signal, threshold);
31
32     std::cout << "Processed signal: ";
33     displaySignal(signal);
34
35     return 0;
36 }
37

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
cd "/home/rps/sunny/" && g++ sigproc1.cpp -o sigproc1 && "/home/rps/sunny/"sigproc1
rps@rps-virtual-machine:~/sunny$ cd "/home/rps/sunny/" && g++ sigproc1.cpp -o sigproc1 &
Original signal: 1 2 3 4 5
Processed signal: 1 2 3 8 10
rps@rps-virtual-machine:~/sunny$
```

Q. 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.

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

volatile sig_atomic_t alarmCounter = 0;

void alarmHandler(int signum) {
    std::cout << "Received SIGALRM: " << signum << std::endl;
    alarmCounter++;
    alarm(5);
}

void sigintHandler(int signum) {
    std::cout << "Received SIGINT: " << signum << ". Ignoring." <<
std::endl;
    signal(SIGINT, SIG_IGN);
}

void sigtermHandler(int signum) {
    std::cout << "Received SIGTERM: " << signum << ". Exiting." <<
std::endl;
    signal(SIGTERM, SIG_DFL);
    raise(SIGTERM);
}

int main() {
    signal(SIGALRM, alarmHandler);
    signal(SIGINT, sigintHandler);
    signal(SIGTERM, sigtermHandler);

    alarm(5);

    while (true) {
        std::cout << "Program running... Press Ctrl+C to send SIGINT. Send
SIGTERM to terminate." << std::endl;
```

```

        std::this_thread::sleep_for(std::chrono::seconds(1));
    }

    return 0;
}

```

```

file Edit Selection View Go Run Terminal Help
EXPLORER
SUNNY
  alarm
  alarm.cpp
  block
  block.cpp
  choice.sh
  echo.sh
  filechecker.sh
  fun.sh
  greet.sh
  greeting.sh
  hello_world.sh
  ignore
  ignore.cpp
  interrupted
  interrupted.cpp
  kill.sh
  ks.txt
  last
  last.cpp
  multiple
  multiple.cpp
  processsignal
  processsignal.cpp
  ques2
  ques2.cpp
multiple.cpp  ignore.cpp  last.cpp  processsignal.cpp  sigproc.cpp  sigproc1.cpp  alarm.cpp x
alarm.cpp > sigtermHandler(int)
2  #include <csignal>
3  #include <unistd.h>
4  #include <chrono>
5  #include <thread>
6
7  volatile sig_atomic_t alarmCounter = 0;
8
9  void alarmHandler(int signum) {
10     std::cout << "Received SIGALRM: " << signum << std::endl;
11     alarmCounter++;
12     alarm(5);
13 }
14
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
Code + -
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Received SIGALRM: 14
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Received SIGALRM: 14
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
^CProgram running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.
Program running... Press Ctrl+C to send SIGINT. Send SIGTERM to terminate.

```

```

#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>

```

```

#define PORT 8080

```

```

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);
std::cout << "Message from client: " << buffer << std::endl;
send(new_socket, hello, strlen(hello), 0);
std::cout << "Hello message sent\n";
close(new_socket);
close(server_fd);
return 0;
}

```

```

#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

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) {
        std::cout << "Socket creation error" << std::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) {
        std::cout << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cout << "Connection Failed" << std::endl;
        return -1;
    }
    send(sock, hello, strlen(hello), 0);
    std::cout << "Hello message sent\n";
    valread = read(sock, buffer, 1024);
    std::cout << "Message from server: " << buffer << std::endl;
    close(sock);
    return 0;
}

```

## Chat client and server code in c++

### server

```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>

#define PORT 8080

void handle_client(int client_socket) {
    char buffer[1024] = {0};
    while (true) {

        int valread = read(client_socket, buffer, 1024);
        if (valread <= 0) {
            std::cout << "Client disconnected" << std::endl;
            break;
        }
        std::cout << "Client: " << buffer << std::endl;

        std::string server_message;
        std::cout << "Server: ";
        std::getline(std::cin, server_message);
        send(client_socket, server_message.c_str(), server_message.length(), 0);
    }
    close(client_socket);
}

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
}
```

```

    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);

    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);
    }
    std::cout << "Server listening on port " << PORT << std::endl;

    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        exit(EXIT_FAILURE);
    }
    std::cout << "Accepted a new connection" << std::endl;

    handle_client(new_socket);

    close(server_fd);
    return 0;
}

```

## client

```

#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

```

```

int main() {
    int sock = 0;
    struct sockaddr_in serv_addr;
    char buffer[1024] = {0};

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cout << "Socket creation error" << std::endl;
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cout << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cout << "Connection Failed" << std::endl;
        return -1;
    }

    while (true) {
        std::string client_message;
        std::cout << "Client: ";
        std::getline(std::cin, client_message);
        send(sock, client_message.c_str(), client_message.length(), 0);

        int valread = read(sock, buffer, 1024);
        if (valread <= 0) {
            std::cout << "Server disconnected" << std::endl;
            break;
        }
        std::cout << "Server: " << buffer << std::endl;
        memset(buffer, 0, sizeof(buffer));
    }

    close(sock);
    return 0;
}

```