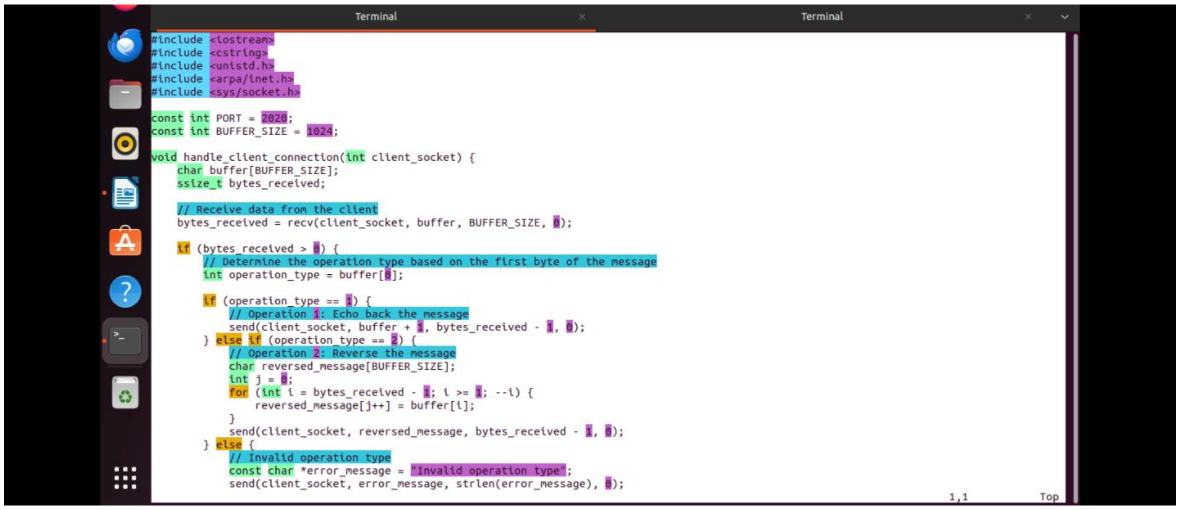
Day – 6 LSP : Test

Assignment Task -1

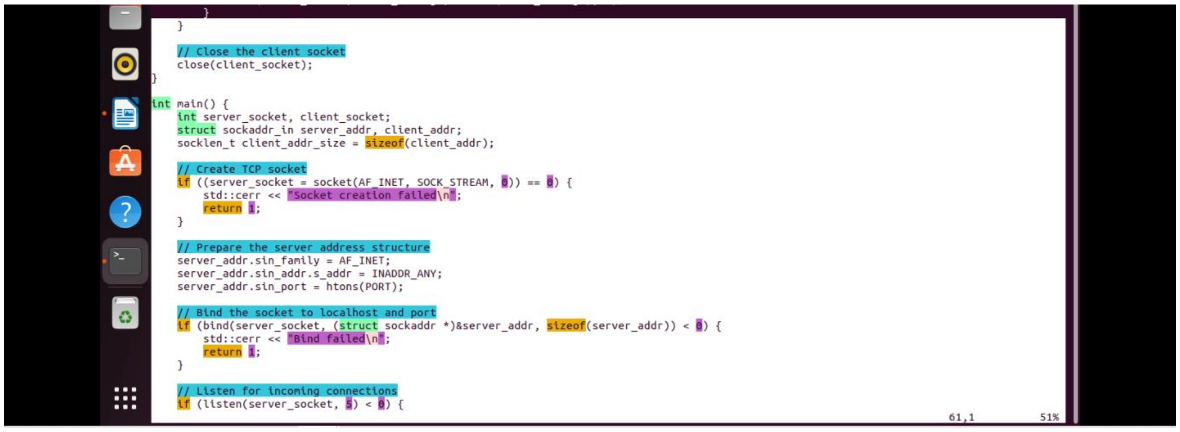
1. TCP Server with Custom Protocol:

Implement a TCP server that:
Binds to port 2020. Listens for incoming connections.
Implements a simple custom protocol where:
The first byte of the message indicates the type of operation (e.g., 1 for echo, 2 for reverse).
For operation type 1, the server echoes the message back.
For operation type 2, the server sends back the reversed message.
Closes the connection and terminates.

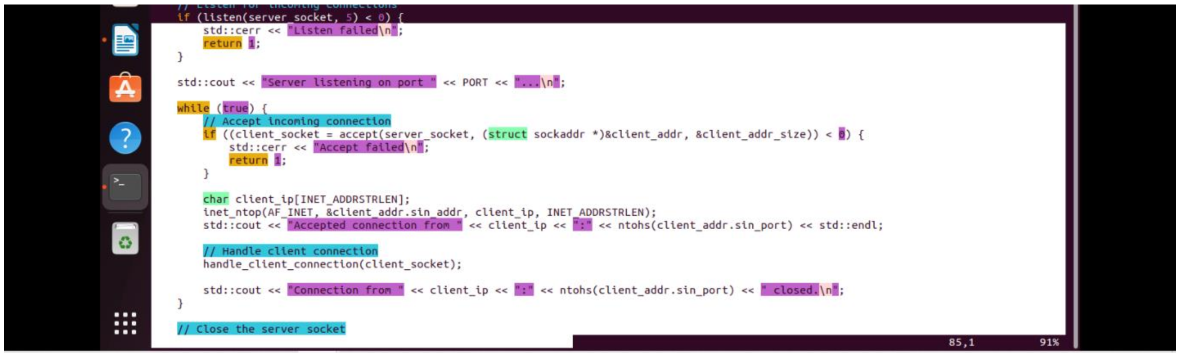
Tcp client



```
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

const int PORT = 2020;
const int BUFFER_SIZE = 1024;

void handle_client_connection(int client_socket) {
    char buffer[BUFFER_SIZE];
    ssize_t bytes_received;

    // Receive data from the client
    bytes_received = recv(client_socket, buffer, BUFFER_SIZE, 0);

    if (bytes_received > 0) {
        // Determine the operation type based on the first byte of the message
        int operation_type = buffer[0];

        if (operation_type == 1) {
            // Operation 1: Echo back the message
            send(client_socket, buffer + 1, bytes_received - 1, 0);
        } else if (operation_type == 2) {
            // Operation 2: Reverse the message
            char reversed_message[BUFFER_SIZE];
            int j = 0;
            for (int i = bytes_received - 1; i >= 1; --i) {
                reversed_message[j++] = buffer[i];
            }
            send(client_socket, reversed_message, bytes_received - 1, 0);
        } else {
            // Invalid operation type
            const char *error_message = "Invalid operation type";
            send(client_socket, error_message, strlen(error_message), 0);
                                                                              1,1           Top
```

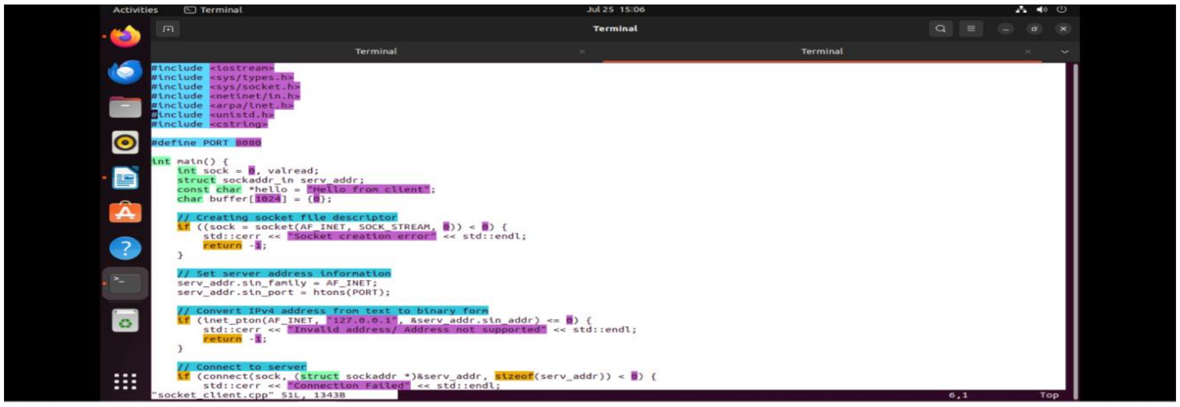


```
        }
    }

    // Close the client socket
    close(client_socket);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_size = sizeof(client_addr);

    // Create TCP socket
    if ((server_socket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        std::cerr << "Socket creation failed\n";
        return 1;
    }

    // Prepare the server address structure
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket to localhost and port
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        std::cerr << "Bind failed\n";
        return 1;
    }

    // Listen for incoming connections
    if (listen(server_socket, 5) < 0) {
                                                                              61,1          51%
```



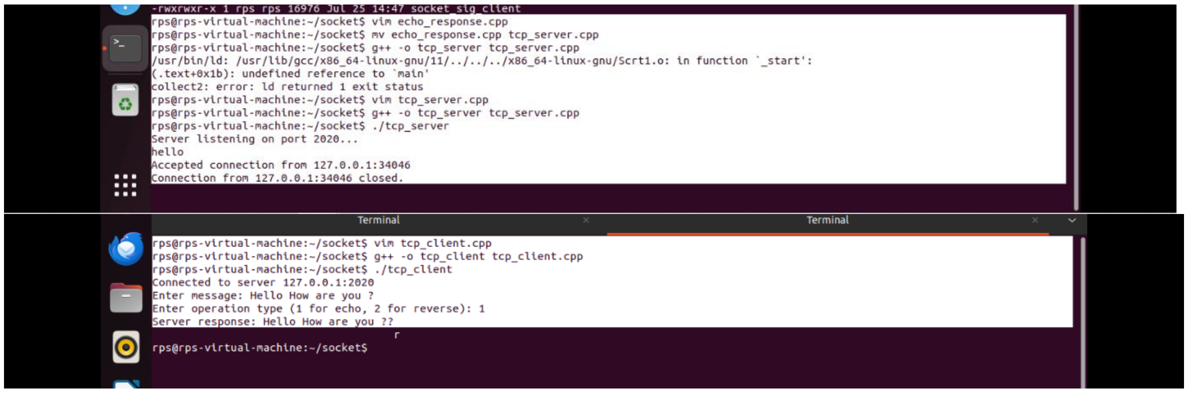
```
    if (listen(server_socket, 5) < 0) {
        std::cerr << "Listen failed\n";
        return 1;
    }

    std::cout << "Server listening on port " << PORT << "...\n";

    while (true) {
        // Accept incoming connection
        if ((client_socket = accept(server_socket, (struct sockaddr *)&client_addr, &client_addr_size)) < 0) {
            std::cerr << "Accept failed\n";
            return 1;
        }

        char client_ip[INET_ADDRSTRLEN];
        inet_ntop(AF_INET, &client_addr.sin_addr, client_ip, INET_ADDRSTRLEN);
        std::cout << "Accepted connection from " << client_ip << ":" << ntohs(client_addr.sin_port) << std::endl;

        // Handle client connection
        handle_client_connection(client_socket);

        std::cout << "Connection from " << client_ip << ":" << ntohs(client_addr.sin_port) << " closed.\n";
    }

    // Close the server socket
                                                                                85,1          91%
```

Tcp_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, valread;
    struct sockaddr_in serv_addr;
    const char *hello = "Hello from client";
    char buffer[1024] = {0};

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

    // Set server address information
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed" << std::endl;
"socket_client.cpp" 51L, 1343B                                                  6,1           Top
```

Execution :



```
-rwxrwxr-x 1 rps rps 16976 Jul 25 14:47 socket_sig_client
rps@rps-virtual-machine:~/socket$ vim echo_response.cpp
rps@rps-virtual-machine:~/socket$ mv echo_response.cpp tcp_server.cpp
rps@rps-virtual-machine:~/socket$ g++ -o tcp_server tcp_server.cpp
/usr/bin/ld: /usr/lib/gcc/x86_64-linux-gnu/11/../../../x86_64-linux-gnu/Scrt1.o: in function `_start':
(.text+0x1b): undefined reference to `main'
collect2: error: ld returned 1 exit status
rps@rps-virtual-machine:~/socket$ vim tcp_server.cpp
rps@rps-virtual-machine:~/socket$ g++ -o tcp_server tcp_server.cpp
rps@rps-virtual-machine:~/socket$ ./tcp_server
Server listening on port 2020...
hello
Accepted connection from 127.0.0.1:34046
Connection from 127.0.0.1:34046 closed.
```

```
rps@rps-virtual-machine:~/socket$ vim tcp_client.cpp
rps@rps-virtual-machine:~/socket$ g++ -o tcp_client tcp_client.cpp
rps@rps-virtual-machine:~/socket$ ./tcp_client
Connected to server 127.0.0.1:2020
Enter message: Hello How are you ?
Enter operation type (1 for echo, 2 for reverse): 1
Server response: Hello How are you ??
                                         r
rps@rps-virtual-machine:~/socket$
```

Assignment Task -2

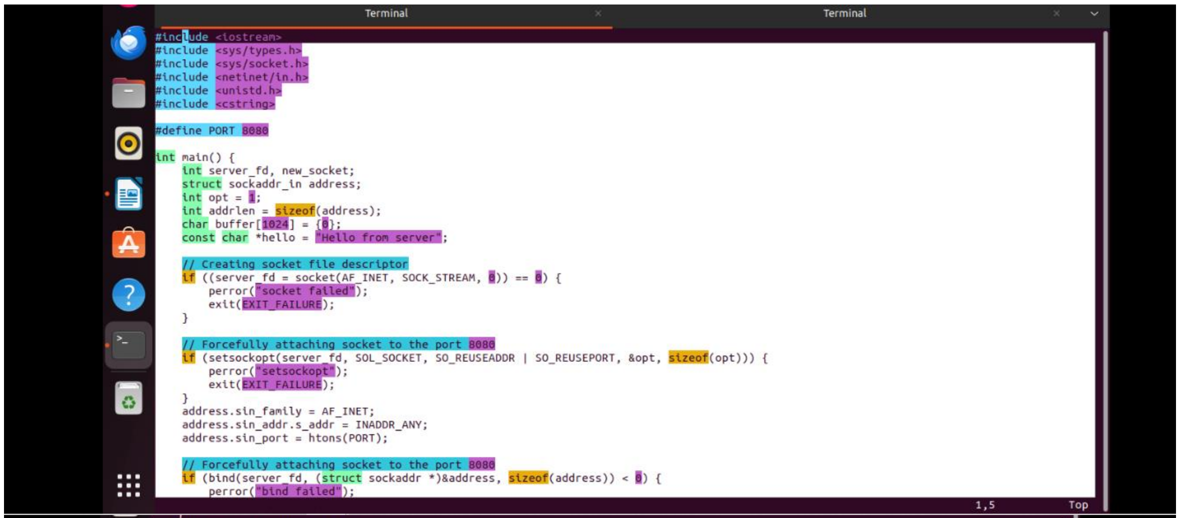
Objective:

Create a C++ application that combines signal handling and socket programming to manage network communication while gracefully handling interruptions (e.g., SIGINT for program termination). The application should be capable of sending and receiving messages over a network while responding appropriately to system signals.

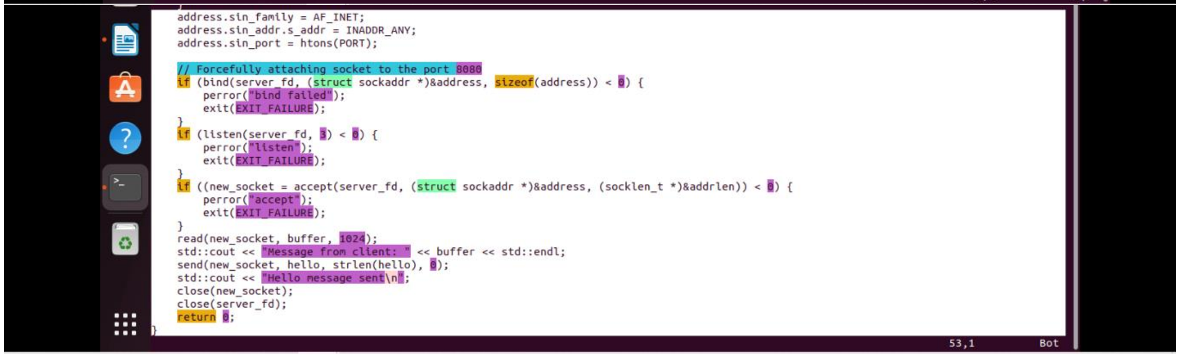
Requirements:

1. **Socket Programming:**
   Implement a TCP server that listens for incoming connections on a specified port.
   Implement a TCP client that connects to the server and exchanges messages.

   a. **TCP Server**

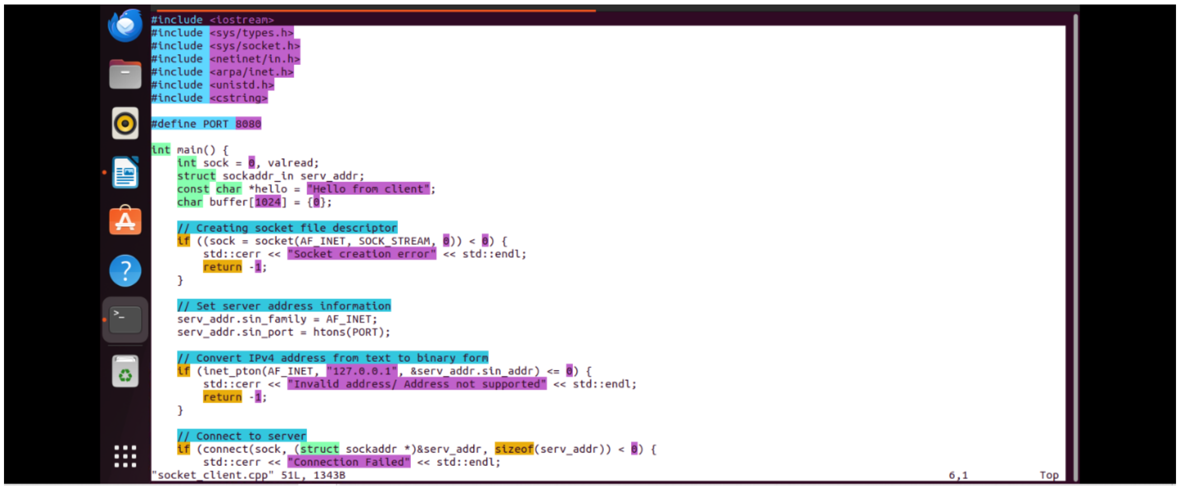


```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>

#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");
```



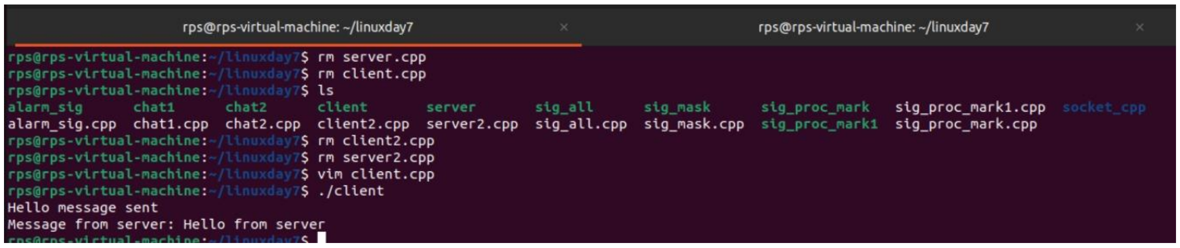
```
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // 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;
}
```

b. TCP Client

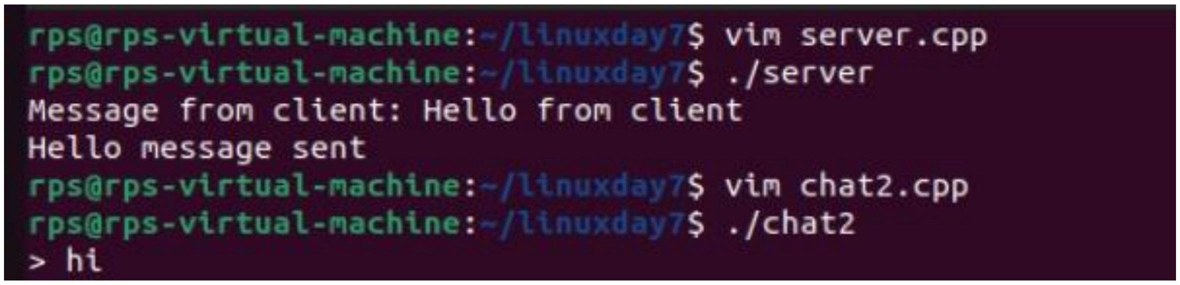


```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>

#define PORT 8080

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    const char *hello = "Hello from client";
    char buffer[1024] = {0};

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

    // Set server address information
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 address from text to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        return -1;
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed" << std::endl;
"socket_client.cpp" 51L, 1343B                                    6,1          Top
```

Execution :



```
rps@rps-virtual-machine: ~/linuxday7                    rps@rps-virtual-machine: ~/linuxday7
rps@rps-virtual-machine:~/linuxday7$ rm server.cpp
rps@rps-virtual-machine:~/linuxday7$ rm client.cpp
rps@rps-virtual-machine:~/linuxday7$ ls
alarm_sig      chat1      chat2      client       server       sig_all      sig_mask      sig_proc_mark    sig_proc_mark1.cpp  socket_cpp
alarm_sig.cpp  chat1.cpp  chat2.cpp  client2.cpp  server2.cpp  sig_all.cpp  sig_mask.cpp  sig_proc_mark1   sig_proc_mark.cpp
rps@rps-virtual-machine:~/linuxday7$ rm client2.cpp
rps@rps-virtual-machine:~/linuxday7$ rm server2.cpp
rps@rps-virtual-machine:~/linuxday7$ vim client.cpp
rps@rps-virtual-machine:~/linuxday7$ ./client
Hello message sent
Message from server: Hello from server
rps@rps-virtual-machine:~/linuxday7$
```



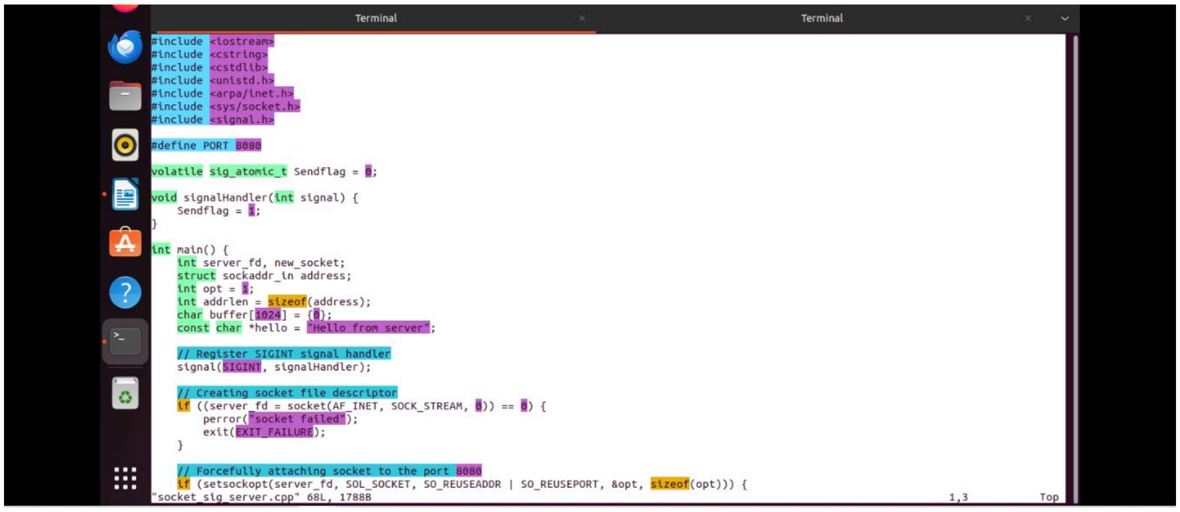
```
rps@rps-virtual-machine:~/linuxday7$ vim server.cpp
rps@rps-virtual-machine:~/linuxday7$ ./server
Message from client: Hello from client
Hello message sent
rps@rps-virtual-machine:~/linuxday7$ vim chat2.cpp
rps@rps-virtual-machine:~/linuxday7$ ./chat2
> hi
```

2. **Signal Handling:**
   Implement signal handlers for SIGINT (Ctrl+C) and SIGTERM to gracefully shut down the server and client. Ensure that the program can handle interruptions without crashing : or leaving resources unfreed.

   i. Data Exchange
   The client should be able to send a message to the server. The server should echo the received message back to the client.

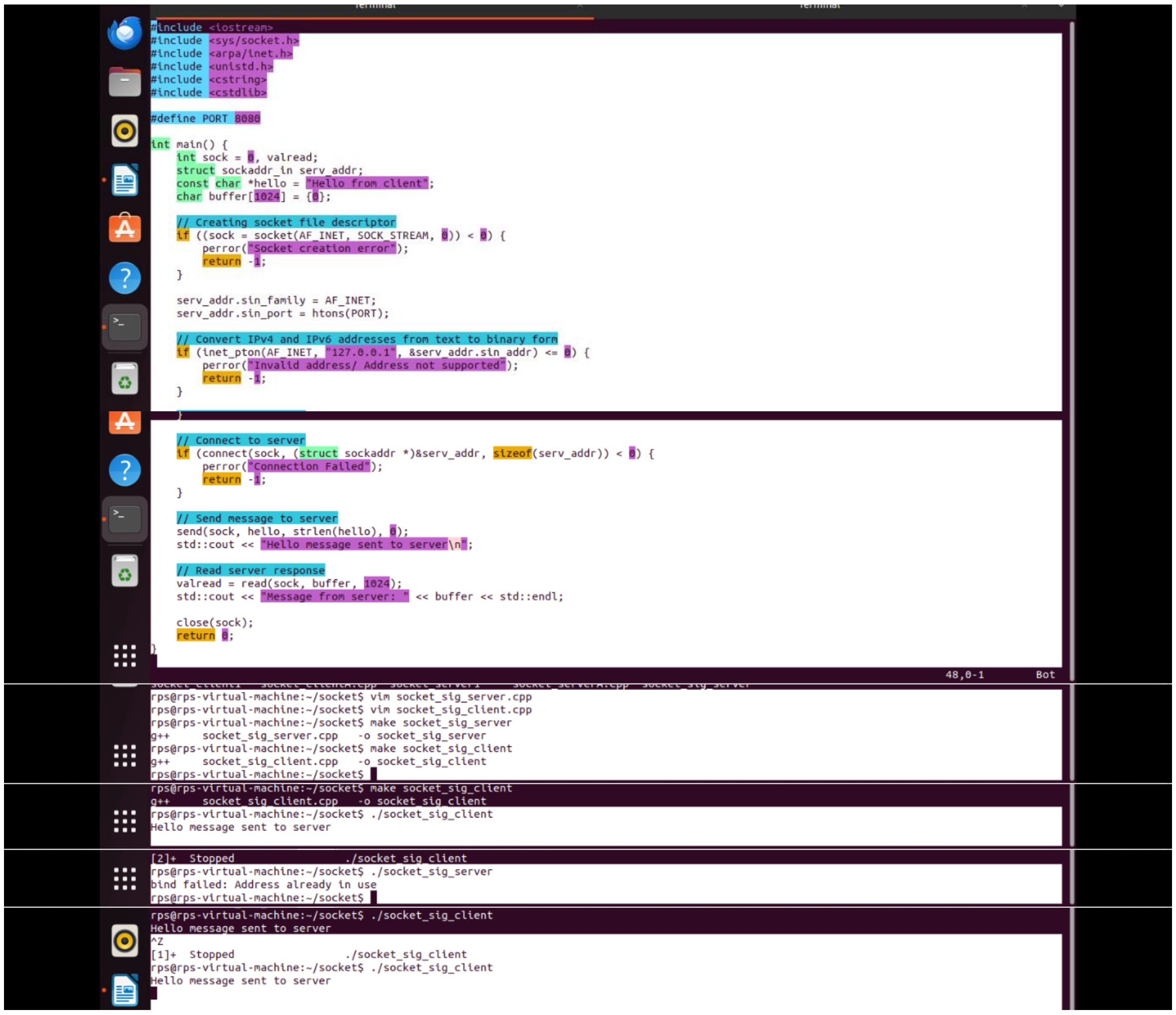
   ii. **Graceful Shutdown:**
   When the server receives a SIGINT or SIGTERM signal, it should close all active connections and free resources before terminating. When the client receives a SIGINT or SIGTERM signal, it should inform the server before terminating.

Server side :



```
#include <iostream>
#include <cstring>
#include <cstdlib>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>
#include <signal.h>

#define PORT 8080

volatile sig_atomic_t Sendflag = 0;

void signalHandler(int signal) {
    Sendflag = 1;
}

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";

    // Register SIGINT signal handler
    signal(SIGINT, signalHandler);

    // 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))) {
"socket_sig_server.cpp" 68L, 1788B                                          1,3           Top
```

Client side :



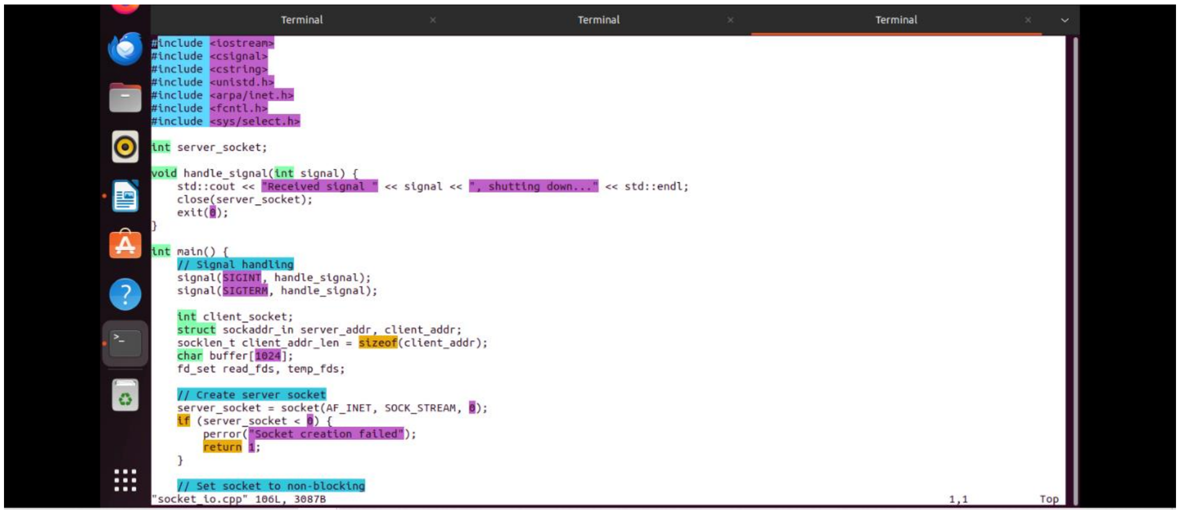
```
#include <iostream>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <cstdlib>

#define PORT 8080

int main() {
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    const char *hello = "Hello from client";
    char buffer[1024] = {0};

    // Creating socket file descriptor
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        perror("Socket creation error");
        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) {
        perror("Invalid address/ Address not supported");
        return -1;
    }
    }
    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("Connection Failed");
        return -1;
    }

    // Send message to server
    send(sock, hello, strlen(hello), 0);
    std::cout << "Hello message sent to server\n";

    // Read server response
    valread = read(sock, buffer, 1024);
    std::cout << "Message from server: " << buffer << std::endl;

    close(sock);
    return 0;
}
                                                                            48,0-1        Bot
```

```
rps@rps-virtual-machine:~/socket$ vim socket_sig_server.cpp
rps@rps-virtual-machine:~/socket$ vim socket_sig_client.cpp
rps@rps-virtual-machine:~/socket$ make socket_sig_server
g++     socket_sig_server.cpp   -o socket_sig_server
rps@rps-virtual-machine:~/socket$ make socket_sig_client
g++     socket_sig_client.cpp   -o socket_sig_client
rps@rps-virtual-machine:~/socket$
rps@rps-virtual-machine:~/socket$ make socket_sig_client
g++     socket_sig_client.cpp   -o socket_sig_client
rps@rps-virtual-machine:~/socket$ ./socket_sig_client
Hello message sent to server

[2]+  Stopped                 ./socket_sig_client
rps@rps-virtual-machine:~/socket$ ./socket_sig_server
bind failed: Address already in use
rps@rps-virtual-machine:~/socket$
rps@rps-virtual-machine:~/socket$ ./socket_sig_client
Hello message sent to server
^Z
[1]+  Stopped                 ./socket_sig_client
rps@rps-virtual-machine:~/socket$ ./socket_sig_client
Hello message sent to server
```

Problem 3: Asynchronous I/O with Signals

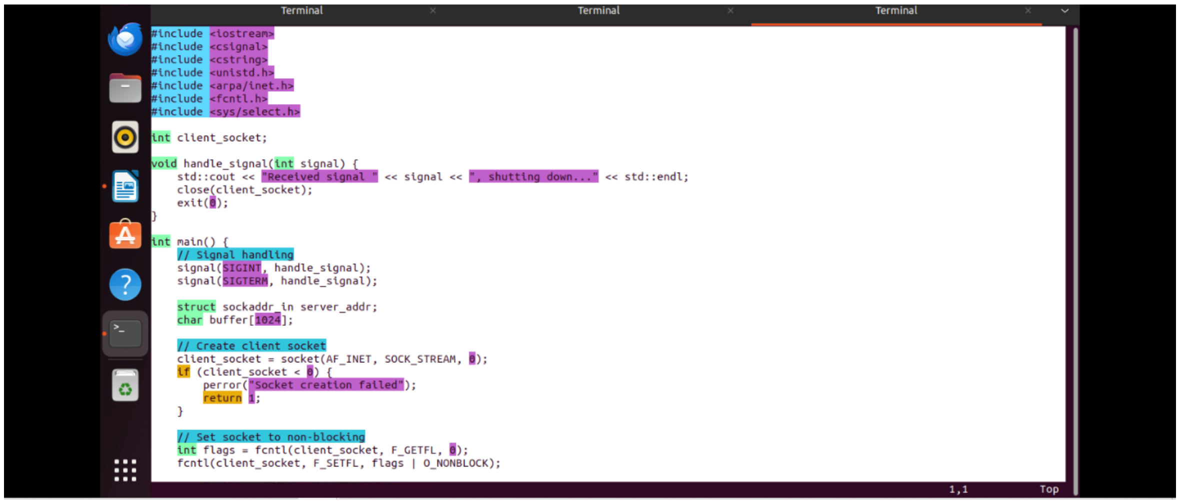
Create a C++ program that uses asynchronous I/O operations for reading from and writing to a socket. Implement signal handling to manage program interruptions and ensure that all pending I/O operations are completed or properly canceled before the program exits.

a. Server-side

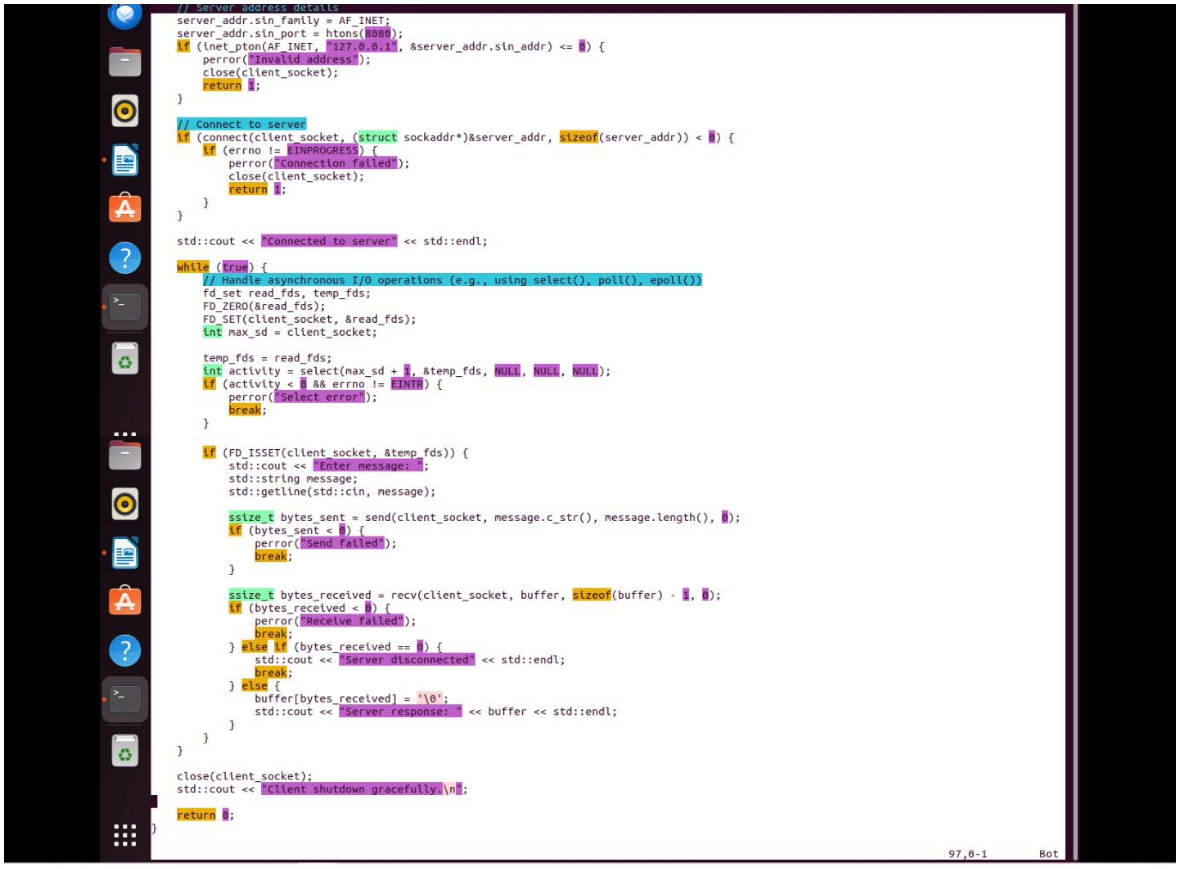


```
#include <iostream>
#include <csignal>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/select.h>

int server_socket;

void handle_signal(int signal) {
    std::cout << "Received signal " << signal << ", shutting down..." << std::endl;
    close(server_socket);
    exit(0);
}

int main() {
    // Signal handling
    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    int client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_addr_len = sizeof(client_addr);
    char buffer[1024];
    fd_set read_fds, temp_fds;

    // Create server socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set socket to non-blocking
"socket_io.cpp" 106L, 3087B                                         1,1           Top
```

b. Client – side

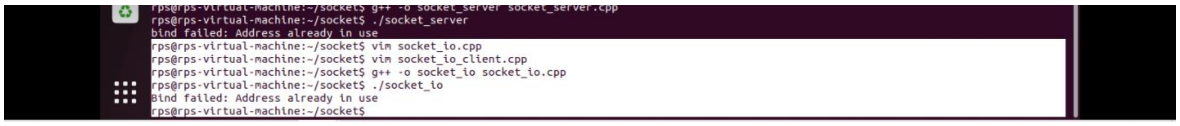


```
#include <iostream>
#include <csignal>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <fcntl.h>
#include <sys/select.h>

int client_socket;

void handle_signal(int signal) {
    std::cout << "Received signal " << signal << ", shutting down..." << std::endl;
    close(client_socket);
    exit(0);
}

int main() {
    // Signal handling
    signal(SIGINT, handle_signal);
    signal(SIGTERM, handle_signal);

    struct sockaddr_in server_addr;
    char buffer[1024];

    // Create client socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Set socket to non-blocking
    int flags = fcntl(client_socket, F_GETFL, 0);
    fcntl(client_socket, F_SETFL, flags | O_NONBLOCK);
                                                                    1,1           Top
```

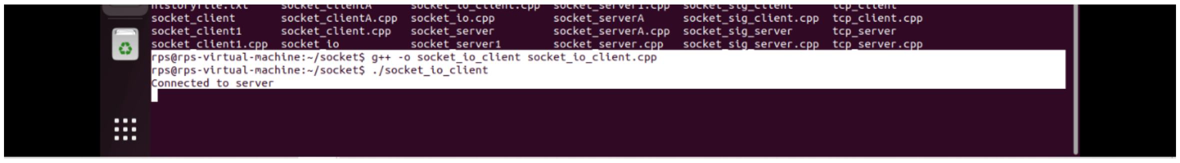


```
// Server address details
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(8080);
if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
    perror("Invalid address");
    close(client_socket);
    return 1;
}

// Connect to server
if (connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
    if (errno != EINPROGRESS) {
        perror("Connection failed");
        close(client_socket);
        return 1;
    }
}

std::cout << "Connected to server" << std::endl;

while (true) {
    // Handle asynchronous I/O operations (e.g., using select(), poll(), epoll())
    fd_set read_fds, temp_fds;
    FD_ZERO(&read_fds);
    FD_SET(client_socket, &read_fds);
    int max_sd = client_socket;

    temp_fds = read_fds;
    int activity = select(max_sd + 1, &temp_fds, NULL, NULL, NULL);
    if (activity < 0 && errno != EINTR) {
        perror("Select error");
        break;
    }

    if (FD_ISSET(client_socket, &temp_fds)) {
        std::cout << "Enter message: ";
        std::string message;
        std::getline(std::cin, message);

        ssize_t bytes_sent = send(client_socket, message.c_str(), message.length(), 0);
        if (bytes_sent < 0) {
            perror("Send failed");
            break;
        }

        ssize_t bytes_received = recv(client_socket, buffer, sizeof(buffer) - 1, 0);
        if (bytes_received < 0) {
            perror("Receive failed");
            break;
        } else if (bytes_received == 0) {
            std::cout << "Server disconnected" << std::endl;
            break;
        } else {
            buffer[bytes_received] = '\0';
            std::cout << "Server response: " << buffer << std::endl;
        }
    }
}

close(client_socket);
std::cout << "Client shutdown gracefully.\n";

return 0;
}
                                                                97,0-1        Bot
```

Execution :



```
rps@rps-virtual-machine:~/socket$ g++ -o socket_server socket_server.cpp
rps@rps-virtual-machine:~/socket$ ./socket_server
bind failed: Address already in use
rps@rps-virtual-machine:~/socket$ vim socket_io.cpp
rps@rps-virtual-machine:~/socket$ vim socket_io_client.cpp
rps@rps-virtual-machine:~/socket$ g++ -o socket_io socket_io.cpp
rps@rps-virtual-machine:~/socket$ ./socket_io
Bind failed: Address already in use
rps@rps-virtual-machine:~/socket$
```



```
historyfile.txt    socket_clientA      socket_io_client.cpp  socket_server1.cpp  socket_sig_client      tcp_client
socket_client      socket_clientA.cpp  socket_io.cpp         socket_serverA      socket_sig_client.cpp  tcp_client.cpp
socket_client1     socket_client.cpp   socket_server         socket_serverA.cpp  socket_sig_server      tcp_server
socket_client1.cpp socket_io           socket_server1        socket_server.cpp   socket_sig_server.cpp  tcp_server.cpp
rps@rps-virtual-machine:~/socket$ g++ -o socket_io_client socket_io_client.cpp
rps@rps-virtual-machine:~/socket$ ./socket_io_client
Connected to server
```