

DATE :- 25-07-2024

1. Write a code where when user put CTRL+C message delivered to server.

Client.cpp

```
rps@rps-virtual-machine:~/sockets$ vim client.cpp
rps@rps-virtual-machine:~/sockets$ make client
g++    client.cpp    -o client
rps@rps-virtual-machine:~/sockets$ ./client
Running... Press Ctrl+C to interrupt.
^C Ctrl+C pressed! Sending message to server...
Message sent. Exiting.
Server response: Message received
```

```

rps@rps-virtual-machine:~/sockets$ cat client.cpp
#include <iostream>
#include <csignal>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

void signalHandler(int signum) {
    std::cout << "Ctrl+C pressed! Sending message to server..." << std::endl;
    int sock = 0;
    struct sockaddr_in serv_addr;
    const char* message = "User pressed Ctrl+C";
    char buffer[1024] = {0};
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "Socket creation error" << std::endl;
        exit(EXIT_FAILURE);
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(8080);
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        std::cerr << "Invalid address/ Address not supported" << std::endl;
        exit(EXIT_FAILURE);
    }
    if (connect(sock, (struct sockaddr*)&serv_addr, sizeof(serv_addr)) < 0) {
        std::cerr << "Connection Failed" << std::endl;
        exit(EXIT_FAILURE);
    }
    send(sock, message, strlen(message), 0);
    std::cout << "Message sent. Exiting." << std::endl;
    read(sock, buffer, 1024);
    std::cout << "Server response: " << buffer << std::endl;
    close(sock);
    exit(signum);
}

```

```

int main() {
    signal(SIGINT, signalHandler);
    std::cout << "Running... Press Ctrl+C to interrupt." << std::endl;
    while (true) {
        pause();
    }
    return 0;
}
rps@rps-virtual-machine:~/sockets$

```

Server.cpp

```
rps@rps-virtual-machine:~/sockets$ vim server.cpp
rps@rps-virtual-machine:~/sockets$ make server
g++      server.cpp  -o server
rps@rps-virtual-machine:~/sockets$ ./server
Server listening on port 8080
Received message: User pressed Ctrl+C
^C
rps@rps-virtual-machine:~/sockets$
```

```
rps@rps-virtual-machine:~/sockets$ cat server.cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

int main() {
    int server_fd, new_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    const char* response = "Message received";

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        std::cerr << "Socket creation error" << std::endl;
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt)))
        std::cerr << "setsockopt error" << std::endl;
        exit(EXIT_FAILURE);
    }
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(8080);

    // Forcefully attaching socket to the port 8080
    if (bind(server_fd, (struct sockaddr*)&address, sizeof(address)) < 0) {
        std::cerr << "Bind failed" << std::endl;
        exit(EXIT_FAILURE);
    }
}
```

```

    if (listen(server_fd, 3) < 0) {
        std::cerr << "Listen error" << std::endl;
        exit(EXIT_FAILURE);
    }
    std::cout << "Server listening on port 8080" << std::endl;

    while (true) {
        if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {
            std::cerr << "Accept error" << std::endl;
            exit(EXIT_FAILURE);
        }
        read(new_socket, buffer, 1024);
        std::cout << "Received message: " << buffer << std::endl;
        send(new_socket, response, strlen(response), 0);
        close(new_socket);
    }

    return 0;
}
rps@rps-virtual-machine:~/sockets$

```

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:

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.

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.

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.

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.

Client:

```
Signal (2) received. Shutting down client gracefully.
rps@rps-virtual-machine:~/sockets$ ./client3
Message sent
Message from server: Hello from server
^C
Signal (2) received. Shutting down client gracefully.
rps@rps-virtual-machine:~/sockets$ vim client3.cpp
rps@rps-virtual-machine:~/sockets$ make client3
g++      client3.cpp      -o client3
```

```
Signal (2) received. Shutting down client gracefully...
rps@rps-virtual-machine:~/sockets$ cat client3.cpp
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <cstring>
#include <csignal>

#define PORT 8080

int sock = 0;
bool running = true;

void signalHandler(int signum) {
    std::cout << "\nSignal (" << signum << ") received. Shutting down client gracefully...\n";
    running = false;
    if (sock >= 0) {
        send(sock, "Client shutting down", 20, 0);
        close(sock);
    }
    exit(signum);
}

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

    signal(SIGINT, signalHandler);
    signal(SIGTERM, signalHandler);

    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        std::cerr << "\nSocket creation error\n";
    }
}
```

```

if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
    std::cerr << "\nSocket creation error\n";
    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::cerr << "\nInvalid address/ Address not supported\n";
    return -1;
}

if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
    std::cerr << "\nConnection Failed\n";
    return -1;
}

send(sock, message, strlen(message), 0);
std::cout << "Message sent\n";
read(sock, buffer, 1024);
std::cout << "Message from server: " << buffer << std::endl;

while (running) {
    // Keep the client running to handle the signal
    pause();
}

close(sock);
std::cout << "Client shutdown complete.\n";
return 0;
}
rps@rps-virtual-machine:~/sockets$

```

Server:

```

rps@rps-virtual-machine:~/sockets$ vim ser3.cpp
rps@rps-virtual-machine:~/sockets$ make ser3
g++    ser3.cpp    -o ser3
rps@rps-virtual-machine:~/sockets$ ./ser3
Server listening on port 8080
Message from client: Hello from client
Hello message sent
^C
Signal (2) received. Shutting down server gracefully...
rps@rps-virtual-machine:~/sockets$ vim ser3.cpp

```

```
rps@rps-virtual-machine:~/sockets$ cat ser3.cpp
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <cstring>
#include <csignal>

#define PORT 8080

int server_fd, new_socket;
bool running = true;

void signalHandler(int signum) {
    std::cout << "\nSignal (" << signum << ") received. Shutting down server gracefully...\n";
    running = false;
    if (new_socket >= 0) {
        close(new_socket);
    }
    if (server_fd >= 0) {
        close(server_fd);
    }
    exit(signum);
}

int main() {
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);
    char buffer[1024] = {0};
    const char *hello = "Hello from server";

    signal(SIGINT, signalHandler);
    signal(SIGTERM, 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))) {
    perror("setsockopt");
    exit(EXIT_FAILURE);
}
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Binding 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);
}

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

while (running) {
    if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        if (running) {
            perror("accept");
        }
        continue;
    }
    read(new_socket, buffer, 1024);

```

```

        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);
        new_socket = -1;
    }

    close(server_fd);
    std::cout << "Server shutdown complete.\n";
    return 0;
}
rps@rps-virtual-machine:~/sockets$

```

3. Client with Timeout:

Create a TCP client that: Connects to a server at port 3030. Sends a message to the server. Implements a timeout mechanism to handle cases where the server does not respond within a specified time. Receives and prints the response message if available. Closes the socket and terminates.

Client:


```
rps@rps-virtual-machine:~/sockets/today$ vim c2.cpp
rps@rps-virtual-machine:~/sockets/today$ vim c2.cpp
rps@rps-virtual-machine:~/sockets/today$ make c2
g++      c2.cpp      -o c2
rps@rps-virtual-machine:~/sockets/today$ ./c2
Connected to server
Received message from server: Hello, Client!
```

Code:

```
#include <iostream>

#include <cstring> // For memset

#include <unistd.h> // For close

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/select.h>

#include <fcntl.h> // For fcntl

#define SERVER_PORT 3030

#define SERVER_IP "127.0.0.1" // Localhost for testing

#define BUFFER_SIZE 1024

#define TIMEOUT_SEC 5

int main() {

    // Step 1: Create a socket

    int sock = socket(AF_INET, SOCK_STREAM, 0);

    if (sock < 0) {

        std::cerr << "Socket creation failed\n";

        return 1;

    }
```

```

int flags = fcntl(sock, F_GETFL, 0);

fcntl(sock, F_SETFL, flags | O_NONBLOCK)

sockaddr_in server_address;    // Step 2: Connect to the server

memset(&server_address, 0, sizeof(server_address));

server_address.sin_family = AF_INET;

server_address.sin_port = htons(SERVER_PORT);

inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr);

if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {

    if (errno != EINPROGRESS) {

        std::cerr << "Connection failed\n";

        close(sock);

        return 1;

    }

}

fd_set write_fds;    // Step 3: Implement a timeout mechanism

struct timeval timeout;

int select_result;

FD_ZERO(&write_fds);

FD_SET(sock, &write_fds);

timeout.tv_sec = TIMEOUT_SEC;

timeout.tv_usec = 0;

select_result = select(sock + 1, nullptr, &write_fds, nullptr, &timeout);

if (select_result == -1) {

    std::cerr << "Select error\n";

    close(sock);

```

```

    return 1;
} else if (select_result == 0) {

    std::cerr << "Connection timed out\n";

    close(sock);

    return 1;
} else if (FD_ISSET(sock, &write_fds)) {

    std::cout << "Connected to server\n";

}

const char* message = "Hello, Server!";    // Step 4: Send a message to the server

ssize_t bytes_sent = send(sock, message, strlen(message), 0);

if (bytes_sent < 0) {

    std::cerr << "Send failed\n";

    close(sock);

    return 1;
} char buffer[BUFFER_SIZE];

memset(buffer, 0, BUFFER_SIZE);

ssize_t bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);

if (bytes_received < 0) {

    std::cerr << "Receive failed\n";

    close(sock);

    return 1;
} else if (bytes_received == 0) {

    std::cout << "Server closed the connection\n";

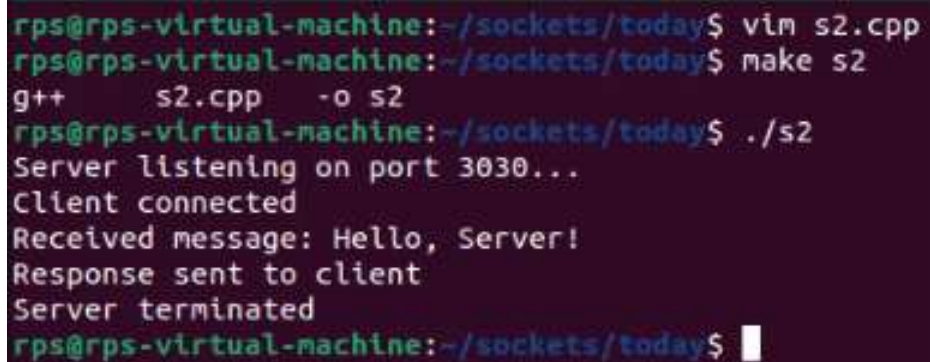
} else {

    std::cout << "Received message from server: " << buffer << "\n";

```

```
}  
  
close(sock);  
  
return 0;  
  
}
```

Server:



A terminal window with a dark purple background and green text. The prompt is 'rps@rps-virtual-machine:~/sockets/today\$'. The user enters 'vim s2.cpp', then 'make s2', which runs 'g++ s2.cpp -o s2'. Then the user enters './s2', which outputs: 'Server listening on port 3030...', 'Client connected', 'Received message: Hello, Server!', 'Response sent to client', and 'Server terminated'. The prompt returns.

```
rps@rps-virtual-machine:~/sockets/today$ vim s2.cpp  
rps@rps-virtual-machine:~/sockets/today$ make s2  
g++ s2.cpp -o s2  
rps@rps-virtual-machine:~/sockets/today$ ./s2  
Server listening on port 3030...  
Client connected  
Received message: Hello, Server!  
Response sent to client  
Server terminated  
rps@rps-virtual-machine:~/sockets/today$
```

Code:

```
#include <iostream>  
  
#include <cstring> // For memset  
  
#include <unistd.h> // For close  
  
#include <sys/types.h>  
  
#include <sys/socket.h>  
  
#include <netinet/in.h>  
  
#include <arpa/inet.h>  
  
#define PORT 3030  
  
#define BUFFER_SIZE 1024  
  
int main() {  
  
    int server_fd = socket(AF_INET, SOCK_STREAM, 0);  
  
    if (server_fd < 0) {  
  
        std::cerr << "Socket creation failed\n";
```



```

    return 1; }

sockaddr_in server_address;

memset(&server_address, 0, sizeof(server_address));

server_address.sin_family = AF_INET;

server_address.sin_addr.s_addr = INADDR_ANY;

server_address.sin_port = htons(PORT);

if (bind(server_fd, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {

    std::cerr << "Bind failed\n";

    close(server_fd);

    return 1; }

if (listen(server_fd, 5) < 0) {

    std::cerr << "Listen failed\n";

    close(server_fd);

    return 1; }

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

sockaddr_in client_address;

socklen_t client_address_len = sizeof(client_address);

int client_fd = accept(server_fd, (struct sockaddr*)&client_address, &client_address_len);

if (client_fd < 0) {

    std::cerr << "Accept failed\n";

    close(server_fd);

    return 1; }

std::cout << "Client connected\n";

char buffer[BUFFER_SIZE];

memset(buffer, 0, BUFFER_SIZE);

```

```

ssize_t bytes_received = recv(client_fd, buffer, BUFFER_SIZE, 0);

if (bytes_received < 0) {

    std::cerr << "Receive failed\n";

    close(client_fd);

    close(server_fd);

    return 1;

} else if (bytes_received == 0) {

    std::cout << "Client disconnected\n";

} else {

    std::cout << "Received message: " << buffer << "\n";

    const char* response = "Hello, Client!";

    ssize_t bytes_sent = send(client_fd, response, strlen(response), 0);

    if (bytes_sent < 0) {

        std::cerr << "Send failed\n";

        close(client_fd);

        close(server_fd);

        return 1;

    } std::cout << "Response sent to client\n";

} close(client_fd);

close(server_fd);

std::cout << "Server terminated\n";

return 0;

}

```

4. CP Echo Server:

Implement a TCP server that:

Binds to port 9090.

Listens for incoming connections.

Accepts a connection from a client.

Receives a message from the client and echoes the same message back to the client.

Closes the connection and terminates.

Client:

```
#include <iostream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <cerrno>

#include <arpa/inet.h> // For inet_pton

int main() {

    int sock_fd;

    struct sockaddr_in server_addr;

    char buffer[1024] = {0};

    const char *message = "Hello, Server!";

    sock_fd = socket(AF_INET, SOCK_STREAM, 0);

    if (sock_fd < 0) {

        perror("Socket creation failed");

        return 1;

    } server_addr.sin_family = AF_INET;

    server_addr.sin_port = htons(9090);

    if (inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr) <= 0) {
```

```
    perror("Invalid address or Address not supported");

    close(sock_fd);

    return 1; }

if (connect(sock_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {

    perror("Connection failed");

    close(sock_fd);

    return 1; }

if (send(sock_fd, message, strlen(message), 0) < 0) {

    perror("Send failed");

    close(sock_fd);

    return 1; }

std::cout << "Message sent to server: " << message << std::endl;

ssize_t valread = read(sock_fd, buffer, sizeof(buffer) - 1);

if (valread < 0) {

    perror("Read failed");

    close(sock_fd);

    return 1; }

buffer[valread] = '\0';

std::cout << "Server echoed: " << buffer << std::endl;

close(sock_fd);

return 0;

}
```



```
rps@rps-virtual-machine:~/sockets/today$ vim c3.cpp
rps@rps-virtual-machine:~/sockets/today$ make c3
g++      c3.cpp  -o c3
rps@rps-virtual-machine:~/sockets/today$ ./c3
Message sent to server: Hello, Server!
Server echoed: Hello, Server!
rps@rps-virtual-machine:~/sockets/today$
```

Server:

```
#include <iostream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

#include <cerrno>

int main() {

    int server_fd, client_fd;

    struct sockaddr_in server_addr, client_addr;

    socklen_t addr_len = sizeof(client_addr);

    char buffer[1024] = {0};

    server_fd = socket(AF_INET, SOCK_STREAM, 0);

    if (server_fd < 0) {

        perror("Socket creation failed");

        return 1; }

    server_addr.sin_family = AF_INET;

    server_addr.sin_addr.s_addr = INADDR_ANY;

    server_addr.sin_port = htons(9090);

    if (bind(server_fd, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {

        perror("Bind failed");
```

```

    close(server_fd);

    return 1;

} if (listen(server_fd, 1) < 0) {

    perror("Listen failed");

    close(server_fd);

    return 1; }

std::cout << "Waiting for connections on port 9090...\n";

client_fd = accept(server_fd, (struct sockaddr*)&client_addr, &addr_len);

if (client_fd < 0) {

    perror("Accept failed");

    close(server_fd);

    return 1; }

std::cout << "Client connected\n";

ssize_t valread = read(client_fd, buffer, sizeof(buffer) - 1);

if (valread < 0) {

    perror("Read failed");

    close(client_fd);

    close(server_fd);

    return 1; }

buffer[valread] = '\0';

std::cout << "Received message: " << buffer << std::endl;

if (send(client_fd, buffer, valread, 0) < 0) {

    perror("Send failed");

    close(client_fd);

    close(server_fd);

```

```

    return 1; }

close(client_fd);

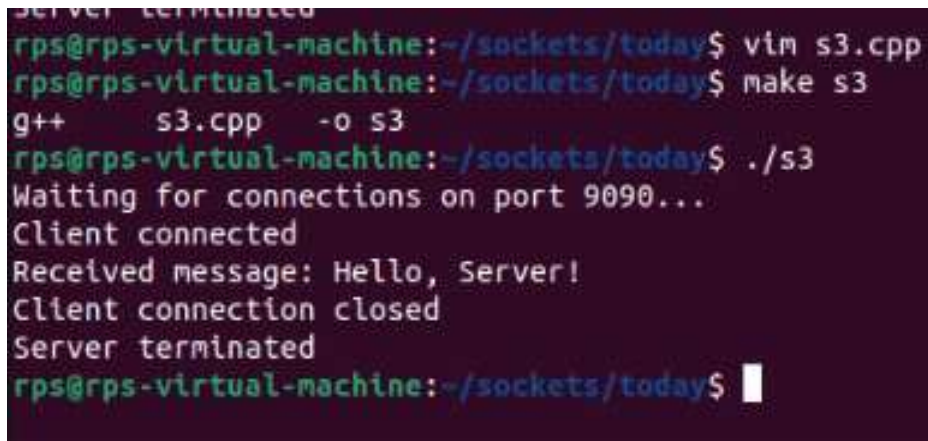
std::cout << "Client connection closed\n";

close(server_fd);

std::cout << "Server terminated\n";

return 0; }

```



```

server terminated
rps@rps-virtual-machine:~/sockets/today$ vim s3.cpp
rps@rps-virtual-machine:~/sockets/today$ make s3
g++      s3.cpp      -o s3
rps@rps-virtual-machine:~/sockets/today$ ./s3
Waiting for connections on port 9090...
Client connected
Received message: Hello, Server!
Client connection closed
Server terminated
rps@rps-virtual-machine:~/sockets/today$

```

5. UDP Client-Server Communication:

Create a UDP server that: Binds to port 7070.

Receives a message from a client.

Sends a response message back to the client.

Closes the socket and terminates.

Create a UDP client that:

Sends a message to the server on port 7070.

Receives and prints the response message from the server.

Closes the socket and terminates.

CLIENT:

```

rps@rps-virtual-machine:~/sockets/today$ vim c5.cpp
rps@rps-virtual-machine:~/sockets/today$ make c5
g++      c5.cpp      -o c5
rps@rps-virtual-machine:~/sockets/today$ ./c5
Message sent to server: Hello, UDP Server!
Server response: Message received
rps@rps-virtual-machine:~/sockets/today$

```

Server:

```

rps@rps-virtual-machine:~/sockets/today$ vim s5.cpp
rps@rps-virtual-machine:~/sockets/today$ make s5
g++      s5.cpp      -o s5
rps@rps-virtual-machine:~/sockets/today$ ./s5
UDP server is running on port 7070
Received message: Hello, UDP Server!
rps@rps-virtual-machine:~/sockets/today$ █

```

6. Implement a TCP server that:

Binds to port 4040.

Listens for incoming connections.

Uses select() to handle multiple client connections.

Receives a message from each client and sends a response back.

Closes the connections and terminates.

Client:

```

rps@rps-virtual-machine:~/sockets/today$ vim c4.cpp
rps@rps-virtual-machine:~/sockets/today$ make c4
g++      c4.cpp      -o c4
rps@rps-virtual-machine:~/sockets/today$ ./c4
Connected to server
Receive failed
rps@rps-virtual-machine:~/sockets/today$ █

```

```
#include <iostream>
```

```
#include <cstring> // For memset
```

```
#include <unistd.h> // For close
```

```
#include <sys/types.h>
```



```

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/select.h>

#include <fcntl.h> // For fcntl

#define SERVER_PORT 4040

#define SERVER_IP "127.0.0.1" // Localhost for testing

#define BUFFER_SIZE 1024

#define TIMEOUT_SEC 5

int main() {

    int sock = socket(AF_INET, SOCK_STREAM, 0);

    if (sock < 0) {

        std::cerr << "Socket creation failed\n";

        return 1; }

    int flags = fcntl(sock, F_GETFL, 0);

    fcntl(sock, F_SETFL, flags | O_NONBLOCK);

    sockaddr_in server_address;

    memset(&server_address, 0, sizeof(server_address));

    server_address.sin_family = AF_INET;

    server_address.sin_port = htons(SERVER_PORT);

    inet_pton(AF_INET, SERVER_IP, &server_address.sin_addr);

    if (connect(sock, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {

        if (errno != EINPROGRESS) {

            std::cerr << "Connection failed\n";

            close(sock);

```

```

        return 1;    }
    }

    fd_set write_fds;

    struct timeval timeout;

    int select_result;

    FD_ZERO(&write_fds);

    FD_SET(sock, &write_fds);

    timeout.tv_sec = TIMEOUT_SEC;

    timeout.tv_usec = 0;

    select_result = select(sock + 1, nullptr, &write_fds, nullptr, &timeout);

    if (select_result == -1) {

        std::cerr << "Select error\n";

        close(sock);

        return 1;

    } else if (select_result == 0) {

        std::cerr << "Connection timed out\n";

        close(sock);

        return 1;

    } else if (FD_ISSET(sock, &write_fds)) {

        std::cout << "Connected to server\n"; }

    const char* message = "Hello, Server!";

    ssize_t bytes_sent = send(sock, message, strlen(message), 0);

    if (bytes_sent < 0) {

        std::cerr << "Send failed\n";

        close(sock);

```

```

    return 1; }

char buffer[BUFFER_SIZE];

memset(buffer, 0, BUFFER_SIZE);

ssize_t bytes_received = recv(sock, buffer, BUFFER_SIZE, 0);

if (bytes_received < 0) {

    std::cerr << "Receive failed\n";

    close(sock);

    return 1;

} else if (bytes_received == 0) {

    std::cout << "Server closed the connection\n";

} else {

    std::cout << "Received message from server: " << buffer << "\n"; }

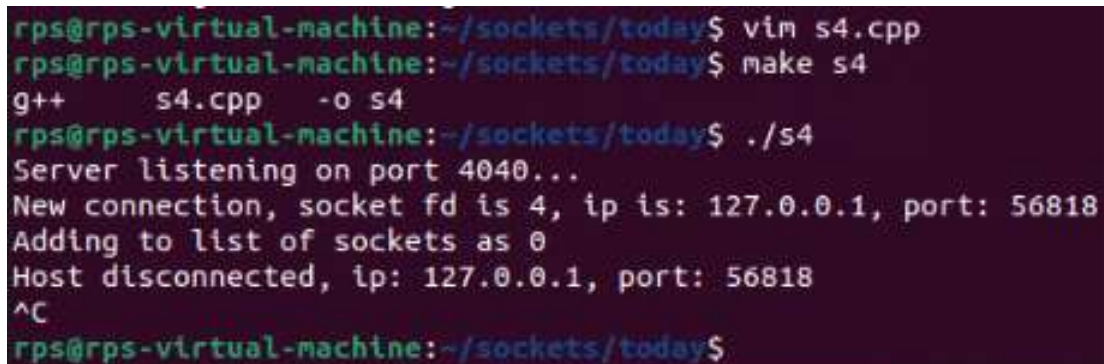
close(sock);

return 0;

}

```

SERVER:



```

rps@rps-virtual-machine:~/sockets/today$ vim s4.cpp
rps@rps-virtual-machine:~/sockets/today$ make s4
g++    s4.cpp    -o s4
rps@rps-virtual-machine:~/sockets/today$ ./s4
Server listening on port 4040...
New connection, socket fd is 4, ip is: 127.0.0.1, port: 56818
Adding to list of sockets as 0
Host disconnected, ip: 127.0.0.1, port: 56818
^C
rps@rps-virtual-machine:~/sockets/today$

```

```
#include <iostream>
```

```
#include <cstring>    // For memset
```

```
#include <unistd.h>    // For close
```

```

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <sys/select.h>

#define PORT 4040

#define BUFFER_SIZE 1024

#define MAX_CLIENTS 10

int main() {

    int server_fd, new_socket, client_socket[MAX_CLIENTS], max_clients = MAX_CLIENTS, activity, i,
    valread, sd;

    int max_sd;

    struct sockaddr_in address;

    char buffer[BUFFER_SIZE];

    int addrlen = sizeof(address);

    for (i = 0; i < max_clients; i++) {

        client_socket[i] = 0;

    } server_fd = socket(AF_INET, SOCK_STREAM, 0);

    if (server_fd < 0) {

        std::cerr << "Socket creation failed\n";

        return 1;

    } 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) {

        std::cerr << "Bind failed\n";

```



```

    close(server_fd);

    return 1; }

if (listen(server_fd, 3) < 0) {

    std::cerr << "Listen failed\n";

    close(server_fd);

    return 1; }

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

fd_set readfds;

while (true) {

    FD_ZERO(&readfds);

    FD_SET(server_fd, &readfds);

    max_sd = server_fd;

    for (i = 0; i < max_clients; i++) {

        sd = client_socket[i];

        if (sd > 0) {

            FD_SET(sd, &readfds);

        }

        if (sd > max_sd) {

            max_sd = sd; }

    }

    activity = select(max_sd + 1, &readfds, NULL, NULL, NULL);

    if ((activity < 0) && (errno != EINTR)) {

        std::cerr << "Select error\n"; }

    if (FD_ISSET(server_fd, &readfds)) {

        int addrlen = sizeof(address);

```

```

if ((new_socket = accept(server_fd, (struct sockaddr*)&address, (socklen_t*)&addrlen)) < 0) {

    std::cerr << "Accept failed\n";

    close(server_fd);

    return 1;

} std::cout << "New connection, socket fd is " << new_socket << ", ip is: "

    << inet_ntoa(address.sin_addr) << ", port: " << ntohs(address.sin_port) << "\n";

for (i = 0; i < max_clients; i++) {

    if (client_socket[i] == 0) {

        client_socket[i] = new_socket;

        std::cout << "Adding to list of sockets as " << i << "\n";

        break;    } }

}

for (i = 0; i < max_clients; i++) {

    sd = client_socket[i];

    if (FD_ISSET(sd, &readfds)) {

        if ((valread = read(sd, buffer, BUFFER_SIZE)) == 0) {

            getpeername(sd, (struct sockaddr*)&address, (socklen_t*)&addrlen);

            std::cout << "Host disconnected, ip: " << inet_ntoa(address.sin_addr) << ", port: " <<
ntohs(address.sin_port) << "\n";

            close(sd);

            client_socket[i] = 0;

        } else {

            buffer[valread] = '\0';

            send(sd, buffer, strlen(buffer), 0);    }

        } }

} close(server_fd);

```

```
    return 0;
}
```

7. TCP Client with Error Handling:

Create a TCP client that:

Connects to a server at port 5050.

Sends a message to the server.

Handles and displays error messages for common issues such as connection failure or data transmission errors.

Receives and prints the response message from the server.

Closes the socket and terminates.

Client:

```
#include <iostream>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <cerrno>

using namespace std;

void tcp_client() {

    int client_socket = socket(AF_INET, SOCK_STREAM, 0);

    if (client_socket == -1) {

        perror("Socket creation failed");

        return;

    } struct sockaddr_in server_address;

    memset(&server_address, 0, sizeof(server_address));
```

```

server_address.sin_family = AF_INET;

server_address.sin_port = htons(5050);

if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {

    perror("Invalid address or Address not supported");

    close(client_socket);

    return;

}

if (connect(client_socket, (struct sockaddr*)&server_address, sizeof(server_address)) < 0) {

    perror("Connection failed");

    close(client_socket);

    return;

} const char *message = "Hello, Server!";

if (send(client_socket, message, strlen(message), 0) < 0) {

    perror("Send failed");

    close(client_socket);

    return; }

cout << "Message sent to server: " << message << endl;

char buffer[1024];

ssize_t valread = read(client_socket, buffer, sizeof(buffer) - 1);

if (valread < 0) {

    perror("Read failed");

    close(client_socket);

    return;

}

buffer[valread] = '\0'; // Null-terminate the received data

```

```

cout << "Server response: " << buffer << endl;

close(client_socket); }

int main() {

    tcp_client();

    return 0;

}

```

```

Server response: message received
rps@rps-virtual-machine:~/sockets/today$ vim c6.cpp
Trash -virtual-machine:~/sockets/today$ make c6
g++ c6.cpp -o c6
rps@rps-virtual-machine:~/sockets/today$ ./c6
Message sent to server: Hello, Server!
Server response: Hello, Server!
rps@rps-virtual-machine:~/sockets/today$

```

Server:

```

rps@rps-virtual-machine:~/sockets/today$ vim s6.cpp
rps@rps-virtual-machine:~/sockets/today$ make s6
g++ s6.cpp -o s6
rps@rps-virtual-machine:~/sockets/today$ ./s6
Server is listening on port 5050...
New connection accepted. Client connected from 127.0.0.1:54898
Message from client: Hello, Server!
rps@rps-virtual-machine:~/sockets/today$ █

```

message from client: hello, server!

rps@rps-virtual-machine:~/sockets/today\$ cat s6.cpp

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

```
using namespace std;
```

```
void tcp_server() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len;
    char buffer[1024];

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

```
    // Prepare server address
    memset(&server_addr, 0, sizeof(server_addr));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(5050);
```

```
    // Bind socket to address
    if (bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr)) == -1)
        perror("Bind failed");
    close(server_socket);
    return;
```

```

        cout << "New connection accepted. Client connected from "
              << inet_ntoa(client_addr.sin_addr)
              << ":" << ntohs(client_addr.sin_port) << endl;

        // Receive message from client
        ssize_t valread = read(client_socket, buffer, sizeof(buffer) - 1);
        if (valread == -1) {
            perror("Read failed");
            close(client_socket);
            close(server_socket);
            return;
        }

        buffer[valread] = '\0';
        cout << "Message from client: " << buffer << endl;

        // Echo message back to client
        if (send(client_socket, buffer, strlen(buffer), 0) == -1) {
            perror("Send failed");
            close(client_socket);
            close(server_socket);
            return;
        }

        // Close client socket
        close(client_socket);

        // Close server socket
        close(server_socket);
    }

    int main() {
        tcp_server();
        return 0;
    }
rps@rps-virtual-machine:~/sockets/today$

```

8. 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.

Client

```

rps@rps-virtual-machine:~/sockets/today$ vim c7.cpp
rps@rps-virtual-machine:~/sockets/today$ make c7
g++    c7.cpp    -o c7
rps@rps-virtual-machine:~/sockets/today$ ./c7
Server response: Hello, Server!

```



```

rps@rps-virtual-machine:~/sockets/today$ cat c7.cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 2020

void sendMessage(int sock, int operationType, const std::string &message) {
    std::string fullMessage = std::string(1, operationType) + message;
    send(sock, fullMessage.c_str(), fullMessage.size(), 0);

    char buffer[1024] = {0};
    int bytesRead = read(sock, buffer, sizeof(buffer));
    if (bytesRead > 0) {
        std::cout << "Server response: " << std::string(buffer + 1, bytesRead - 1) << std::endl;
    }
}

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;
    }
}

```

```

// Send echo message
sendMessage(sock, 1, "Hello, Server!");

// Send reverse message
sendMessage(sock, 2, "Hello, Server!");

close(sock);
return 0;
}
rps@rps-virtual-machine:~/sockets/today$ █

```

Server:

```
rps@rps-virtual-machine:~/sockets/today$  
rps@rps-virtual-machine:~/sockets/today$ vim s7.cpp  
rps@rps-virtual-machine:~/sockets/today$ make s7  
g++      s7.cpp      -o s7  
rps@rps-virtual-machine:~/sockets/today$ ./s7  
Server is listening on port 2020  
^C  
rps@rps-virtual-machine:~/sockets/today$ cat s7.cpp  
#include <iostream>  
#include <cstring>  
#include <unistd.h>  
#include <arpa/inet.h>  
#include <algorithm>  
  
#define PORT 2020  
  
void handleClient(int clientSocket) {  
    char buffer[1024] = {0};  
    int bytesRead = read(clientSocket, buffer, sizeof(buffer));  
  
    if (bytesRead > 0) {  
        int operationType = buffer[0];  
        std::string message(buffer + 1, bytesRead - 1);  
  
        if (operationType == 1) {  
            // Echo the message back  
            send(clientSocket, buffer, bytesRead, 0);  
        } else if (operationType == 2) {  
            // Reverse the message  
            std::reverse(message.begin(), message.end());  
            std::string response = std::string(1, operationType) + message;  
            send(clientSocket, response.c_str(), response.size(), 0);  
        } else {  
            std::cerr << "Unknown operation type: " << operationType << std::endl;  
        }  
    }  
}
```

```

    close(clientSocket);
}

int main() {
    int serverSocket, clientSocket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Creating socket file descriptor
    if ((serverSocket = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("Socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 2020
    if (setsockopt(serverSocket, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT, &opt, sizeof(opt))) {
        perror("setsockopt");
        close(serverSocket);
        exit(EXIT_FAILURE);
    }

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

    // Binding the socket to the port 2020
    if (bind(serverSocket, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        close(serverSocket);
        exit(EXIT_FAILURE);
    }

    // Listening for incoming connections
    if (listen(serverSocket, 3) < 0) {

```

```

// Listening for incoming connections
if (listen(serverSocket, 3) < 0) {
    perror("listen");
    close(serverSocket);
    exit(EXIT_FAILURE);
}

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

while (true) {
    if ((clientSocket = accept(serverSocket, (struct sockaddr *)&address, (socklen_t *)&addrlen)) < 0) {
        perror("accept");
        close(serverSocket);
        exit(EXIT_FAILURE);
    }

    handleClient(clientSocket);
}

return 0;
}
ros@ros-virtual-machine:~/sockets/today$

```