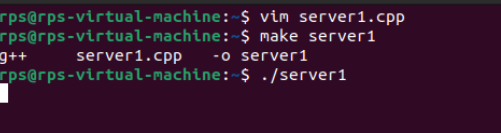
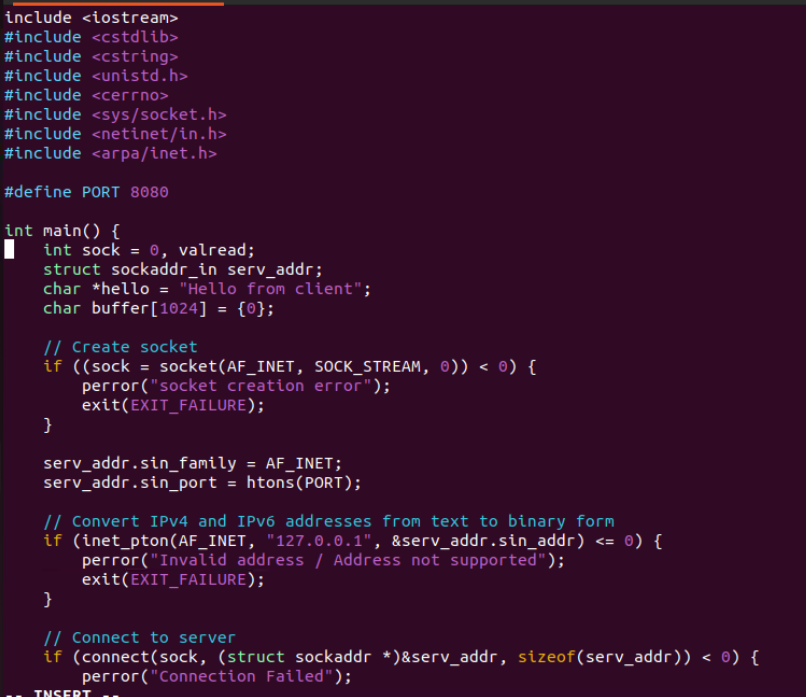
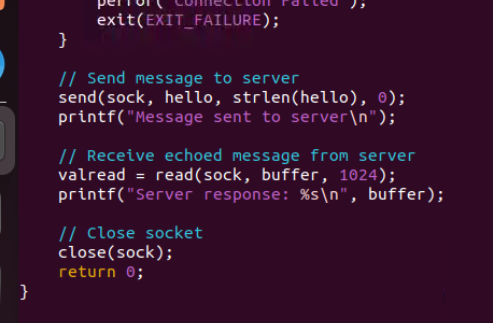
**Design and implement a network service that reliably handles concurrent client connections while ensuring graceful termination in response to external signals (e.g., SIGTERM, SIGINT). The service must maintain data consistency and avoid resource leaks throughout its lifecycle.**

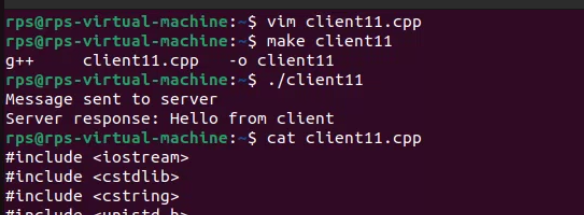
SERVER







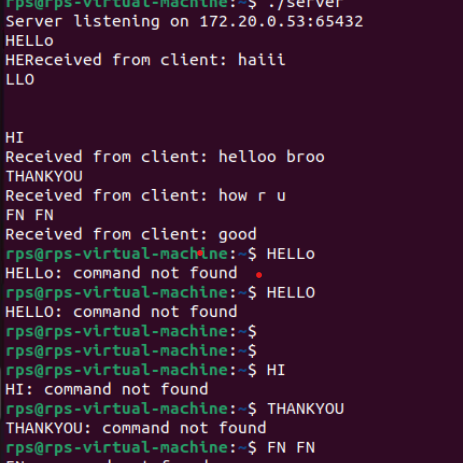
CLINET







**EXCHANGE MESSGAES FORM CLIENT AND SERVER**



**SERVER CODE:**

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Define the server address

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = inet\_addr("172.20.0.53");

address.sin\_port = htons(PORT);

// Bind the socket to the network address and port

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 3) < 0) {

perror("listen");

close(server\_fd);

exit(EXIT\_FAILURE);

}

std::cout << "Server listening on 172.20.0.53:" << PORT << std::endl;

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr )&address, (socklen\_t)&addrlen)) < 0) {

perror("accept");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Communicate with the client

while (true) {

memset(buffer, 0, BUFFER\_SIZE);

int valread = read(new\_socket, buffer, BUFFER\_SIZE);

if (valread <= 0) {

break;

}

std::cout << "Received from client: " << buffer << std::endl;

send(new\_socket, buffer, strlen(buffer), 0);

}

close(new\_socket);

close(server\_fd);

    return 0;

}

**CLIENT CODE:**

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

std::cerr << "Socket creation error" << std::endl;

return -1;

}

// Define the server address

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, "172.20.0.53", &serv\_addr.sin\_addr) <= 0) {

std::cerr << "Invalid address/ Address not supported" << std::endl;

return -1;

}

// Connect to the server

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

std::cerr << "Connection Failed" << std::endl;

return -1;

}

while (true) {

std::cout << "Enter message to send (type 'exit' to close): ";

std::string message;

std::getline(std::cin, message);

if (message == "exit") {

break;

}

send(sock, message.c\_str(), message.length(), 0);

std::cout << "Message sent" << std::endl;

memset(buffer, 0, BUFFER\_SIZE);

int valread = read(sock, buffer, BUFFER\_SIZE);

std::cout << "Received from server: " << buffer << std::endl;

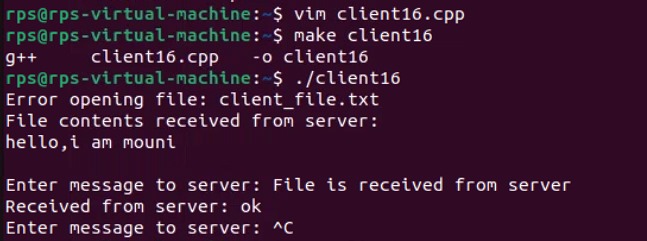
}

close(sock);

    return 0;

}

**how to create a text file and then send from clinet to server and vice versa**

****

**Server code:**

#include <iostream>

#include <fstream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#define PORT 65432

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("socket failed");

exit(EXIT\_FAILURE);

}

// Define the server address

address.sin\_family = AF\_INET;

address.sin\_addr.s\_addr = inet\_addr("172.20.0.53"); // Bind to any available address

address.sin\_port = htons(PORT);

// Bind the socket to the network address and port

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

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

// Accept a connection

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t \*)&addrlen)) < 0) {

perror("accept");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Read the contents of the text file

std::ifstream file("file1.txt");

if (!file) {

perror("File open failed");

close(new\_socket);

close(server\_fd);

exit(EXIT\_FAILURE);

}

std::string file\_contents((std::istreambuf\_iterator<char>(file)), std::istreambuf\_iterator<char>());

file.close();

// Send the contents of the text file to the client

send(new\_socket, file\_contents.c\_str(), file\_contents.size(), 0);

// Communication loop

while (true) {

// Read from client

memset(buffer, 0, BUFFER\_SIZE);

int valread = read(new\_socket, buffer, BUFFER\_SIZE);

if (valread <= 0) {

break;

}

std::cout << "Received from client: " << buffer << std::endl;

// Process the received message (if needed)

// Example: Check if client wants to end communication

if (strcmp(buffer, "quit") == 0) {

std::cout << "Client has requested to quit. Closing connection." << std::endl;

break;

}

// Server's response

std::string server\_response;

std::cout << "Enter message to client: ";

std::getline(std::cin, server\_response);

// Send response back to client

send(new\_socket, server\_response.c\_str(), server\_response.size(), 0);

}

// Close sockets

close(new\_socket);

close(server\_fd);

return 0;

}

**CLIENT CODE:**

#include <iostream>

#include <cstring>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fstream>

#include <vector>

#define PORT 65432

#define BUFFER\_SIZE 1024

void sendFile(int sock, const std::string& filename) {

std::ifstream file(filename, std::ios::binary | std::ios::ate);

if (!file.is\_open()) {

std::cerr << "Error opening file: " << filename << std::endl;

return;

}

std::streamsize file\_size = file.tellg();

file.seekg(0, std::ios::beg);

std::vector<char> buffer(file\_size);

if (file.read(buffer.data(), file\_size)) {

send(sock, buffer.data(), file\_size, 0);

} else {

std::cerr << "Error reading file: " << filename << std::endl;

}

}

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

std::cerr << "Socket creation error" << std::endl;

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

if (inet\_pton(AF\_INET, “172.20.0.53", &serv\_addr.sin\_addr) <= 0) {

std::cerr << "Invalid address or Address not supported" << std::endl;

close(sock);

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

std::cerr << "Connection failed" << std::endl;

close(sock);

return -1;

}

std::string file\_to\_send = "client\_file.txt";

sendFile(sock, file\_to\_send);

std::string file\_contents;

ssize\_t bytes\_received;

while ((bytes\_received = recv(sock, buffer, BUFFER\_SIZE, 0)) > 0) {

file\_contents.append(buffer, bytes\_received);

if (bytes\_received < BUFFER\_SIZE) break;

}

if (bytes\_received < 0) {

std::cerr << "Error receiving file data" << std::endl;

close(sock);

return -1;

}

std::cout << "File contents received from server:" << std::endl;

std::cout << file\_contents << std::endl;

while (true) {

std::string message;

std::cout << "Enter message to server: ";

std::getline(std::cin, message);

send(sock, message.c\_str(), message.size(), 0);

if (message == "quit") {

std::cout << "Ending communication with server." << std::endl;

break;

}

memset(buffer, 0, BUFFER\_SIZE);

ssize\_t valread = recv(sock, buffer, BUFFER\_SIZE, 0);

if (valread > 0) {

std::cout << "Received from server: " << buffer << std::endl;

} else if (valread == 0) {

std::cout << "Server closed the connection." << std::endl;

break;

} else {

std::cerr << "Error receiving data from server" << std::endl;

break;

}

}

close(sock);

return 0;

}