

## **UNIVERSITY OF NAIROBI**

**FACULTY OF SCIENCE AND TECHNOLOGY**

**DEPARTMENT OF COMPUTING AND INFORMATICS**

CSC 322: NETWORK & DISTRIBUTED PROGRAMMING

**GROUP WORK**

## 

**GROUP MEMBERS.**

**WEKULO FIONA ABIGAEL - P15/1905/2020**

**IGADWA DEBI TELOORI - P15/1906/2020**

**MWANGI ENOCH BABU - P15/140432/2020**

**KARANJA SAMUEL MWANGI - P15/1933/2020**

**EMMANUEL MBITHI MUTHIANI - P15/1894/2020**

**Assignment 2**

Develop a client server program which implements a simple calculator. The calculator can perform the following operations (+, -, \*, /) chosen by the user on any two integers supplied by the user.

<https://github.com/swafey-karanja/C-programming./tree/main/Question%201>

# 1.ITERATIVE CONNECTIONLESS.

**Simple Calculator using UDP**

The code provided is a simple calculator program that uses User Datagram Protocol (UDP) to send and receive data between a server and a client.

***Server***

The server program starts by creating a UDP socket using the socket() function. It then binds the socket to an IP address and port using the bind() function. The server then listens for incoming messages from clients using a loop that continuously receives data using the recvfrom() function.

When the server receives a message from a client, it extracts the two numbers and the operator from the message using the sscanf() function, performs the requested calculation, and sends the result back to the client using the sendto() function.

***Client***

The client program starts by creating a UDP socket using the socket() function. It then prompts the user to enter two numbers and select an operator. Once the user inputs the values, the client sends a message to the server using the sendto() function.

The client then waits for a response from the server using the recv() function. Once the client receives the response, it displays the result to the user and prompts the user to perform another operation or exit.

**Conceptual Server Algorithm.**

The server algorithm for this program can be summarized as follows:

1. Create a socket using the socket() system call.
2. Bind the socket to an IP address and port using the bind() system call.
3. Listen for incoming connections using the listen() system call.
4. Accept incoming connections using the accept() system call, which creates a new socket for each client.
5. Read data from the client socket using recv() or recvfrom() system call.
6. Process the received data according to the protocol, which in this case involves performing simple mathematical operations.
7. Send the result of the operation back to the client using send() or sendto() system call.
8. Close the socket using the close() system call when the communication is complete.

In this program, we use recvfrom() and sendto() system calls instead of recv() and send(), because the server is implemented as a UDP server, which is a connectionless protocol. Therefore, we do not need to maintain a connection between the client and server. Instead, we simply send and receive datagrams to and from the client.

Code.

*Server side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

int main() {

// Create a socket

int s\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (s\_socket == -1) {

fprintf(stderr, "Failed to create socket\n");

return 1;

}

// Bind the socket to an IP address and port

struct sockaddr\_in s\_address;

memset(&s\_address, 0, sizeof(s\_address));

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

s\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(s\_socket, (struct sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

fprintf(stderr, "Failed to bind socket\n");

return 1;

}

// Display the port the server is running on

printf("Server running on port %d\n", ntohs(s\_address.sin\_port));

// Read messages from clients

char buffer[1024];

while (1) {

// Accept a request from a client

struct sockaddr\_in client\_address;

memset(&client\_address, 0, sizeof(client\_address));

socklen\_t client\_address\_size = sizeof(client\_address);

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recvfrom(s\_socket, buffer, sizeof(buffer), 0,

(struct sockaddr\*)&client\_address, &client\_address\_size);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from client\n");

continue;

}

// Log a message when a client sends a request

printf("Received a request from a client\n");

// Parse the message

int num1, num2;

char op;

sscanf(buffer, "%d %d %c", &num1, &num2, &op);

// Perform the calculation

int result;

switch (op) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

fprintf(stderr, "Invalid operator: %c\n", op);

continue;

}

// Send the result back to the client

char response[1024];

sprintf(response, "%d\n", result);

sendto(s\_socket, response, strlen(response), 0,

(struct sockaddr\*)&client\_address, client\_address\_size);

}

// Close the socket

close(s\_socket);

return 0;

}

*Client side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

int main() {

// Create a socket

int c\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);

if (c\_socket == -1) {

fprintf(stderr, "Failed to create socket\n");

return 1;

}

// Set the server address

struct sockaddr\_in s\_address;

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

// Replace "SERVER\_IP\_ADDRESS" with the IP address of the server computer

inet\_pton(AF\_INET, "127.0.0.1", &s\_address.sin\_addr);

// Send messages to the server

char message[1024];

char buffer[1024];

while (1) {

// Prompt the user to enter two numbers

int num1, num2;

printf("Enter any two numbers:\n");

scanf("%d %d", &num1, &num2);

// Prompt the user to select an operator

char op;

printf("------------SIMPLE CALCULATOR------------\n");

printf("Select an operation to perform:\n");

printf("1. Addition \n");

printf("2. Subtraction \n");

printf("3. Multiplication \n");

printf("4. Division \n");

int choice;

scanf("%d", &choice);

switch (choice) {

case 1:

op = '+';

break;

case 2:

op = '-';

break;

case 3:

op = '\*';

break;

case 4:

op = '/';

break;

default:

fprintf(stderr, "Invalid choice\n");

continue;

}

// Send the message to the server

sprintf(message, "%d %d %c\n", num1, num2, op);

sendto(c\_socket, message, strlen(message), 0,

(struct sockaddr\*)&s\_address, sizeof(s\_address));

// Read the result from the server

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from server\n");

break;

} else if (bytes\_received == 0) {

printf("Server disconnected\n");

break;

}

// Display the result

int result = atoi(buffer);

printf("Result: %d\n", result);

// Prompt the user to perform another operation or exit

printf("Choose next action:\n");

printf("1. Perform another operation\n");

printf("2. Exit\n");

int option;

scanf("%d", &option);

if (option == 2) {

break;

}

}

// Close the socket

close(c\_socket);

return 0;

}

# 2.ITERATIVE CONNECTION-ORIENTED.

In this simple client-server program in C language implementing a calculator,the server program creates a socket, binds it to a specific IP address and port number, listens for incoming connections, accepts a connection from a client, reads messages (in the form of two numbers and an operator) from the client, performs the calculation and sends the result back to the client.

The client program creates a socket, connects to the server using the server's IP address and port number, prompts the user to enter two numbers and select an operator, sends the message to the server, reads the result from the server, and displays it to the user.

The server program uses the following libraries:

* stdio.h: for input/output operations
* string.h: for string manipulation operations
* stdlib.h: for memory allocation and deallocation operations
* sys/socket.h: for socket-related operations
* netinet/in.h: for internet-related operations
* unistd.h: for POSIX standard function definitions

The client program uses the following libraries:

* stdio.h: for input/output operations
* stdlib.h: for memory allocation and deallocation operations
* string.h: for string manipulation operations
* sys/socket.h: for socket-related operations
* netinet/in.h: for internet-related operations
* arpa/inet.h: for internet-related operations
* unistd.h: for POSIX standard function definition.

The communication between the client and server is done over the TCP protocol.

**Conceptual Server Algorithm.**

The server algorithm for the program can be summarized as follows:

1. Create a socket using the socket() function.
2. Bind the socket to a specific IP address and port using the bind() function.
3. Listen for incoming connections using the listen() function.
4. Accept an incoming connection using the accept() function.
5. Receive data from the client using the recv() function.
6. Parse the data received from the client to extract the numbers and the operator.
7. Perform the requested calculation based on the operator and the numbers.
8. Send the result back to the client using the send() function.
9. Repeat steps 5 to 8 until the client disconnects or an error occurs.
10. Close the client and server sockets using the close() function.

In summary, the server listens for incoming connections from clients and performs the requested calculations based on the data received from the client. It then sends the result back to the client and repeats the process until the client disconnects or an error occurs.

Code.

*Server\_side.*

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

int main() {

// Create a socket

int s\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (s\_socket == -1) {

fprintf(stderr, "Failed to create socket\n");

return 1;

}

// Bind the socket to an IP address and port

struct sockaddr\_in s\_address;

memset(&s\_address, 0, sizeof(s\_address));

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

s\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(s\_socket, (struct sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

fprintf(stderr, "Failed to bind socket\n");

return 1;

}

// Display the port the server is running on

printf("Server running on port %d\n", ntohs(s\_address.sin\_port));

// Listen for incoming connections

if (listen(s\_socket, 10) == -1) {

fprintf(stderr, "Failed to listen on socket\n");

return 1;

}

// Accept an incoming connection

struct sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int c\_socket = accept(s\_socket, (struct sockaddr\*)&client\_address, &client\_address\_size);

if (c\_socket == -1) {

fprintf(stderr, "Failed to accept connection\n");

return 1;

}

// Log a message when a client connects

printf("Client connected\n");

// Read messages from the client

char buffer[1024];

while (1) {

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from client\n");

break;

} else if (bytes\_received == 0) {

printf("Client disconnected\n");

break;

}

// Parse the message

int num1, num2;

char op;

sscanf(buffer, "%d %d %c", &num1, &num2, &op);

// Perform the calculation

int result;

switch (op) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

fprintf(stderr, "Invalid operator: %c\n", op);

continue;

}

// Send the result back to the client

char response[1024];

sprintf(response, "%d\n", result);

send(c\_socket, response, strlen(response), 0);

}

// Close the sockets

close(c\_socket);

close(s\_socket);

return 0;

}

*Client\_side.*

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

int main() {

// Create a socket

int c\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (c\_socket == -1) {

fprintf(stderr, "Failed to create socket\n");

return 1;

}

// Connect to the server

struct sockaddr\_in s\_address;

memset(&s\_address, 0, sizeof(s\_address));

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

// Replace "SERVER\_IP\_ADDRESS" with the IP address of the server computer

inet\_pton(AF\_INET, "127.0.0.1", &s\_address.sin\_addr);

if (connect(c\_socket, (struct sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

fprintf(stderr, "Failed to connect to server\n");

return 1;

}

// Send messages to the server

char message[1024];

char buffer[1024];

while (1) {

// Prompt the user to enter two numbers

int num1, num2;

printf("Enter any two numbers:\n");

scanf("%d %d", &num1, &num2);

// Prompt the user to select an operator

char op;

printf("------------SIMPLE CALCULATOR------------\n");

printf("Select an operation to perform:\n");

printf("1. Addition \n");

printf("2. Subtraction \n");

printf("3. Multiplication \n");

printf("4. Division \n");

int choice;

scanf("%d", &choice);

switch (choice) {

case 1:

op = '+';

break;

case 2:

op = '-';

break;

case 3:

op = '\*';

break;

case 4:

op = '/';

break;

default:

fprintf(stderr, "Invalid choice\n");

continue;

}

// Send the message to the server

sprintf(message, "%d %d %c\n", num1, num2, op);

send(c\_socket, message, strlen(message), 0);

// Read the result from the server

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

fprintf(stderr, "Failed to receive data from server\n");

break;

} else if (bytes\_received == 0) {

printf("Server disconnected\n");

break;

}

// Display the result

int result = atoi(buffer);

printf("Result: %d\n", result);

// Prompt the user to perform another operation or exit

printf("Choose next action:\n");

printf("1. Perform another operation\n");

printf("2. Exit\n");

int option;

scanf("%d", &option);

if (option == 2) {

break;

}

}

// Close the socket

close(c\_socket);

return 0;

}

# 3.CONCURRENT CONNECTIONLESS.

This is a client-server program in C++ that allows the client to perform arithmetic operations by sending two numbers and an operator to the server, and the server sends back the result of the operation.

The server listens for incoming connections, and when a client connects, it creates a thread to handle the client. The thread reads the message sent by the client, performs the calculation, and sends the result back to the client.

The client connects to the server and prompts the user to enter two numbers and select an operator. It then sends the numbers and operator to the server and waits for the result. Once the result is received, the client displays it to the user.

Here is an overview of the main functions in the server and client programs:

**Server program:**

* handle\_client: a function that runs in a separate thread for each connected client. It reads the message sent by the client, performs the calculation, and sends the result back to the client.
* main: the main function that sets up the server socket, listens for incoming connections, and creates a thread to handle each connected client.

**Client program:**

* main: the main function that sets up the client socket, connects to the server, prompts the user for input, sends the input to the server, receives the result, and displays it to the user.

Conceptual Server Algorithm.

The server algorithm for the program is:

1. Create a socket using socket() function.
2. Bind the socket to an IP address and port using bind() function.
3. Listen for incoming connections using listen() function.
4. Accept incoming connections using accept() function. This function returns a new socket which is used to communicate with the client.
5. Create a new thread to handle the client using pthread\_create() function.
6. In the thread, read messages from the client using recv() function.
7. Parse the message to extract the two numbers and the operator to perform.
8. Perform the requested calculation.
9. Send the result back to the client using send() function.
10. Close the client socket using close() function.
11. Free the memory allocated for the client data.
12. Repeat steps 4 to 11 for each new client connection.
13. Close the server socket using close() function when the server is no longer needed.

Code.

*Server\_side.*

#include <iostream>

#include <string>

#include <sstream>

#include <cstdlib>

#include <cstring>

#include <pthread.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

struct C\_information {

int c\_socket;

std::string filename;

C\_information(int c\_socket, std::string filename)

: c\_socket(c\_socket), filename(filename) {}

};

void\* handle\_client(void\* arg) {

C\_information\* client\_data = (C\_information\*)arg;

int c\_socket = client\_data->c\_socket;

std::string filename = client\_data->filename;

// Read messages from the client

char buffer[1024];

while (true) {

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

} else if (bytes\_received == 0) {

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

break;

}

// Parse the message

std::istringstream message(buffer);

int num1, num2;

char op;

message >> num1 >> num2 >> op;

// Perform the calculation

int result;

switch (op) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

std::cerr << "Invalid operator: " << op << '\n';

continue;

}

// Send the result back to the client

std::ostringstream response;

response << result << '\n';

send(c\_socket, response.str().c\_str(), response.str().length(), 0);

}

// Close the socket

close(c\_socket);

delete client\_data;

return nullptr;

}

int main() {

// Create a socket

int s\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (s\_socket == -1) {

std::cerr << "Failed to create socket\n";

return 1;

}

// Bind the socket to an IP address and port

sockaddr\_in s\_address;

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

s\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(s\_socket, (sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

std::cerr << "Failed to bind socket\n";

return 1;

}

// Display the port the server is running on

std::cout << "Server running on port " << ntohs(s\_address.sin\_port) << '\n';

// Listen for incoming connections

if (listen(s\_socket, 10) == -1) {

std::cerr << "Failed to listen on socket\n";

return 1;

}

while (true) {

// Accept an incoming connection

sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int c\_socket = accept(s\_socket, (sockaddr\*)&client\_address, &client\_address\_size);

if (c\_socket == -1) {

std::cerr << "Failed to accept connection\n";

continue;

}

// Log a message when a client connects

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

// Create a thread to handle the client

pthread\_t thread;

C\_information\* client\_data = new C\_information(c\_socket, "");

pthread\_create(&thread, nullptr, handle\_client, client\_data);

pthread\_detach(thread);

}

// Close the socket

close(s\_socket);

return 0;

}

*Client\_side.*

#include <iostream>

#include <string>

#include <cstdlib>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

int main() {

// Create a socket

int c\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (c\_socket == -1) {

std::cerr << "Failed to create socket\n";

return 1;

}

// Connect to the server

sockaddr\_in s\_address;

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

// Replace "SERVER\_IP\_ADDRESS" with the IP address of the server computer

inet\_pton(AF\_INET, "127.0.0.1", &s\_address.sin\_addr);

if (connect(c\_socket, (sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

std::cerr << "Failed to connect to server\n";

return 1;

}

// Send messages to the server

std::string message;

char buffer[1024];

while (true) {

// Prompt the user to enter two numbers

int num1, num2;

std::cout << "Enter any two numbers:\n";

std::cin >> num1 >> num2;

// Prompt the user to select an operator

char op;

std::cout << "Select operation to perform:\n";

std::cout << "1. Addition\n";

std::cout << "2. Subtraction\n";

std::cout << "3. Multiplication\n";

std::cout << "4. Division\n";

int choice;

std::cin >> choice;

switch (choice) {

case 1:

op = '+';

break;

case 2:

op = '-';

break;

case 3:

op = '\*';

break;

case 4:

op = '/';

break;

default:

std::cerr << "Invalid choice\n";

continue;

}

// Send the message to the server

message = std::to\_string(num1) + ' ' + std::to\_string(num2) + ' ' + op + '\n';

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

// Read the result from the server

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

break;

} else if (bytes\_received == 0) {

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

break;

}

// Display the result

int result = std::stoi(buffer);

std::cout << "Result: " << result << '\n';

// Prompt the user to perform another operation or exit

std::cout << "Choose next action:\n";

std::cout << "1. Perform another operation\n";

std::cout << "2. Exit the program\n";

int option;

std::cin >> option;

if (option == 2) {

break;

}

}

// Close the socket

close(c\_socket);

return 0;

}

# 

# 4.CONCURRENT CONNECTION-ORIENTED.

This is a client-server program that performs basic arithmetic operations (+, -, \*, /) on two numbers provided by the user. The program uses sockets to communicate between the client and server.

The server creates a socket and binds it to a port. It listens for incoming connections and creates a thread to handle each client. The thread reads the client's message, performs the calculation, and sends the result back to the client. The server runs indefinitely until it is terminated.

The client creates a socket and connects to the server. It prompts the user to enter two numbers and select an operator. It sends the numbers and operator to the server, waits for the result, and displays it to the user. The client runs until the user chooses to exit.

Overall, this program demonstrates the basics of socket programming and multithreading in C++.

Conceptual server algorithm.

The conceptual server algorithm for the program is as follows:

1. Create a socket using the socket() function with the AF\_INET (IPv4) address family, SOCK\_STREAM (TCP) type, and 0 protocol.
2. Bind the socket to an IP address and port using the bind() function with a sockaddr\_in structure that specifies the IP address and port to bind to.
3. Listen for incoming connections using the listen() function.
4. Continuously accept incoming connections using the accept() function. For each new connection, create a new thread to handle the client.
5. In the client thread, read messages from the client using the recv() function. Parse the message to extract two numbers and an operator.
6. Perform the requested calculation and send the result back to the client using the send() function.
7. When the client disconnects or an error occurs, close the client socket and exit the thread.
8. When the server is shutting down, close the listening socket.

Code.

*Server\_side.*

#include <iostream>

#include <string>

#include <sstream>

#include <cstdlib>

#include <cstring>

#include <pthread.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <unistd.h>

void\* handle\_client(void\* arg) {

int c\_socket = \*(int\*)arg;

// Read messages from the client

char buffer[1024];

while (true) {

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from client\n";

break;

} else if (bytes\_received == 0) {

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

break;

}

// Parse the message

std::istringstream message(buffer);

int num1, num2;

char op;

message >> num1 >> num2 >> op;

// Perform the calculation

int result;

switch (op) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

result = num1 / num2;

break;

default:

std::cerr << "Invalid operator: " << op << '\n';

continue;

}

// Send the result back to the client

std::ostringstream response;

response << result << '\n';

send(c\_socket, response.str().c\_str(), response.str().length(), 0);

}

// Close the socket

close(c\_socket);

return nullptr;

}

int main() {

// Create a socket

int s\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (s\_socket == -1) {

std::cerr << "Failed to create socket\n";

return 1;

}

// Bind the socket to an IP address and port

sockaddr\_in s\_address;

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

s\_address.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(s\_socket, (sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

std::cerr << "Failed to bind socket\n";

return 1;

}

// Display the port the server is running on

std::cout << "Server running on port " << ntohs(s\_address.sin\_port) << '\n';

// Listen for incoming connections

if (listen(s\_socket, 10) == -1) {

std::cerr << "Failed to listen on socket\n";

return 1;

}

while (true) {

// Accept an incoming connection

sockaddr\_in client\_address;

socklen\_t client\_address\_size = sizeof(client\_address);

int c\_socket = accept(s\_socket, (sockaddr\*)&client\_address, &client\_address\_size);

if (c\_socket == -1) {

std::cerr << "Failed to accept connection\n";

continue;

}

// Log a message when a client connects

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

// Create a thread to handle the client

pthread\_t thread;

pthread\_create(&thread, nullptr, handle\_client, &c\_socket);

pthread\_detach(thread);

}

// Close the socket

close(s\_socket);

return 0;

}

*Client\_side.*

#include <iostream>

#include <string>

#include <cstdlib>

#include <cstring>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

int main() {

// Create a socket

int c\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (c\_socket == -1) {

std::cerr << "Failed to create socket\n";

return 1;

}

// Connect to the server

sockaddr\_in s\_address;

s\_address.sin\_family = AF\_INET;

s\_address.sin\_port = htons(4444);

// Replace "SERVER\_IP\_ADDRESS" with the IP address of the server computer

inet\_pton(AF\_INET, "127.0.0.1", &s\_address.sin\_addr);

if (connect(c\_socket, (sockaddr\*)&s\_address, sizeof(s\_address)) == -1) {

std::cerr << "Failed to connect to server\n";

return 1;

}

// Send messages to the server

std::string message;

char buffer[1024];

while (true) {

// Prompt the user to enter two numbers

int num1, num2;

std::cout << "Enter two numbers (press enter after each input):\n";

std::cin >> num1 >> num2;

// Prompt the user to select an operator

char op;

std::cout << "Select an operator from the list:\n";

std::cout << "1. Addition (+)\n";

std::cout << "2. Subtraction (-)\n";

std::cout << "3. Multiplication (\*)\n";

std::cout << "4. Division (/)\n";

int choice;

std::cin >> choice;

switch (choice) {

case 1:

op = '+';

break;

case 2:

op = '-';

break;

case 3:

op = '\*';

break;

case 4:

op = '/';

break;

default:

std::cerr << "Invalid choice\n";

continue;

}

// Send the message to the server

message = std::to\_string(num1) + ' ' + std::to\_string(num2) + ' ' + op + '\n';

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

// Read the result from the server

memset(buffer, 0, sizeof(buffer));

int bytes\_received = recv(c\_socket, buffer, sizeof(buffer), 0);

if (bytes\_received == -1) {

std::cerr << "Failed to receive data from server\n";

break;

} else if (bytes\_received == 0) {

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

break;

}

// Display the result

int result = std::stoi(buffer);

std::cout << "Result of operation: " << result << '\n';

// Prompt the user to perform another operation or exit

std::cout << "Choose next action:\n";

std::cout << "1. Perform another operation\n";

std::cout << "2. Exit the program\n";

int option;

std::cin >> option;

if (option == 2) {

break;

}

}

// Close the socket

close(c\_socket);

return 0;

}