Client–Server Communication (UDP) in C++ — Step-by-Step Guide

NAME : UBAID KUNDLIK   ASSIGNMENT : 4   PRN :12310428.   ROLL NO : 52.   DIV : CS-AIML-B

3b) Write the client server programs using UDP Berkeley socket primitives for wired /wireless network for following
a. to say Hello to Each other

b. Calculator (Trigonometry)

This document explains how to build, run, and test simple UDP client–server programs in C++ on macOS using VS Code. It covers:

- Hello message (basic client/server)
- **Calculator (Trigonometry)**

Prerequisites

1. **macOS** (you already have).
2. **Xcode Command Line Tools** (compiler):

```
xcode-select --install
```

1.
2. **Visual Studio Code** (editor) and the **C/C++** extension by Microsoft.
3. (Optional) code command installed in PATH so you can open folders from Terminal.

Project folder & files

Create the project folder and open it in VS Code:

```
mkdir -p ~/projects/networking
cd ~/projects/networking
code .
```

Files you will create:

```
networking/
  ├─ udp_server.cpp   # simple hello message server
  ├─ udp_client.cpp   # simple hello message client
  ├─ udp_calc_server.cpp    #
  ├─ udp_calc_client.cpp    # calculates the the values of sin....
```

---

1) Hello Message (basic UDP)

udp_server.cpp

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

#define PORT 8080
#define BUF_SIZE 1024
```

```
int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Setup server address
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;  // bind to all interfaces
    serverAddr.sin_port = htons(PORT);

    // Bind socket
    if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0)
{
        perror("Bind failed");
        close(sockfd);
        return 1;
    }

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

    // Wait for client
    int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                    (struct sockaddr *)&clientAddr, &addrLen);
    buffer[n] = '\0';
    std::cout << "📩 Client: " << buffer << std::endl;

    // Reply
    const char *hello = "Hello from Server!";
    sendto(sockfd, hello, strlen(hello), 0,
           (struct sockaddr *)&clientAddr, addrLen);
    std::cout << "📤 Reply sent to client\n";

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

udp_client.cpp

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

#define PORT 8080
#define BUF_SIZE 1024

int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Setup server address (localhost)
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {
        perror("Invalid address");
        return 1;
    }

    // Send message
    const char *hello = "Hello from Client!";
    sendto(sockfd, hello, strlen(hello), 0,
           (struct sockaddr *)&servaddr, sizeof(servaddr));
    std::cout << "📤 Message sent to server\n";

    // Receive reply
    socklen_t len = sizeof(servaddr);
    int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                     (struct sockaddr *)&servaddr, &len);
    buffer[n] = '\0';
    std::cout << "📩 Server: " << buffer << std::endl;

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

Compile & run

- Terminal 1 (server):

```
g++ udp_server.cpp -o udp_server && ./udp_server
```

Server output:

```
✅ UDP Server listening on port 8080...
```

- Terminal 2 (client):

```
g++ udp_client.cpp -o udp_client && ./udp_client
```

Client output:

```
📤 Message sent to server
📩 Server: Hello from Server!
```

Server receives and replies

At the same time, server shows:

```
📩 Client: Hello from Client!
📤 Reply sent to client
```

👉 So the two programs exchange **Hello messages** successfully over UDP.

Expected output: server prints client message and client prints server message.

---

b. Calculator (Trigonometry)

udp_calc_server.cpp

```cpp
#include <iostream>
#include <cstring>
#include <cmath>
#include <sstream>
#include <unistd.h>
#include <arpa/inet.h>

#define PORT 8081
#define BUF_SIZE 1024

double deg2rad(double deg) {
    return deg * M_PI / 180.0;
}

int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in serverAddr, clientAddr;
    socklen_t addrLen = sizeof(clientAddr);

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Setup server address
    memset(&serverAddr, 0, sizeof(serverAddr));
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;  // bind to all interfaces
    serverAddr.sin_port = htons(PORT);

    // Bind socket
    if (bind(sockfd, (struct sockaddr *)&serverAddr, sizeof(serverAddr)) < 0)
{
        perror("Bind failed");
        close(sockfd);
        return 1;
    }

    std::cout << "🧮 UDP Calculator Server running on port " << PORT <<
"...\n";

    // Keep listening
    while (true) {
        int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                         (struct sockaddr *)&clientAddr, &addrLen);
        buffer[n] = '\0';

        std::string request(buffer);
        std::string op;
        double angle;
        double result = 0.0;

        // Parse input: e.g., "sin 45"
        std::stringstream ss(request);
        ss >> op >> angle;

        if (op == "sin")
            result = sin(deg2rad(angle));
        else if (op == "cos")
            result = cos(deg2rad(angle));
        else if (op == "tan")
            result = tan(deg2rad(angle));
        else
            result = NAN;

        std::string response = "Result: " + std::to_string(result);
        sendto(sockfd, response.c_str(), response.size(), 0,
               (struct sockaddr *)&clientAddr, addrLen);

        std::cout << "📩 Request: " << request << " → " << response << "\n";
    }

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

udp_calc_client.cpp

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

#define PORT 8081
#define BUF_SIZE 1024

int main() {
    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    // Create UDP socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        return 1;
    }

    // Setup server address (localhost)
    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);

    if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {
        perror("Invalid address");
        return 1;
    }

    while (true) {
        std::string input;
        std::cout << "\nEnter operation (sin/cos/tan) and angle in degrees (or
'exit'): ";
        getline(std::cin, input);

        if (input == "exit") break;

        sendto(sockfd, input.c_str(), input.size(), 0,
               (struct sockaddr *)&servaddr, sizeof(servaddr));

        socklen_t len = sizeof(servaddr);
        int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                         (struct sockaddr *)&servaddr, &len);
        buffer[n] = '\0';

        std::cout << "📩 Server: " << buffer << std::endl;
    }

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

How to run

1. Terminal 1:

```
g++ udp_calc_server.cpp -o udp_calc_server && ./udp_calc_server
```

```
🧮 UDP Calculator Server running on port 8081...
```

1. Terminal 2:

```
g++ udp_calc_client.cpp -o udp_calc.client && ./udp_calc_client
```

The client will prompt:

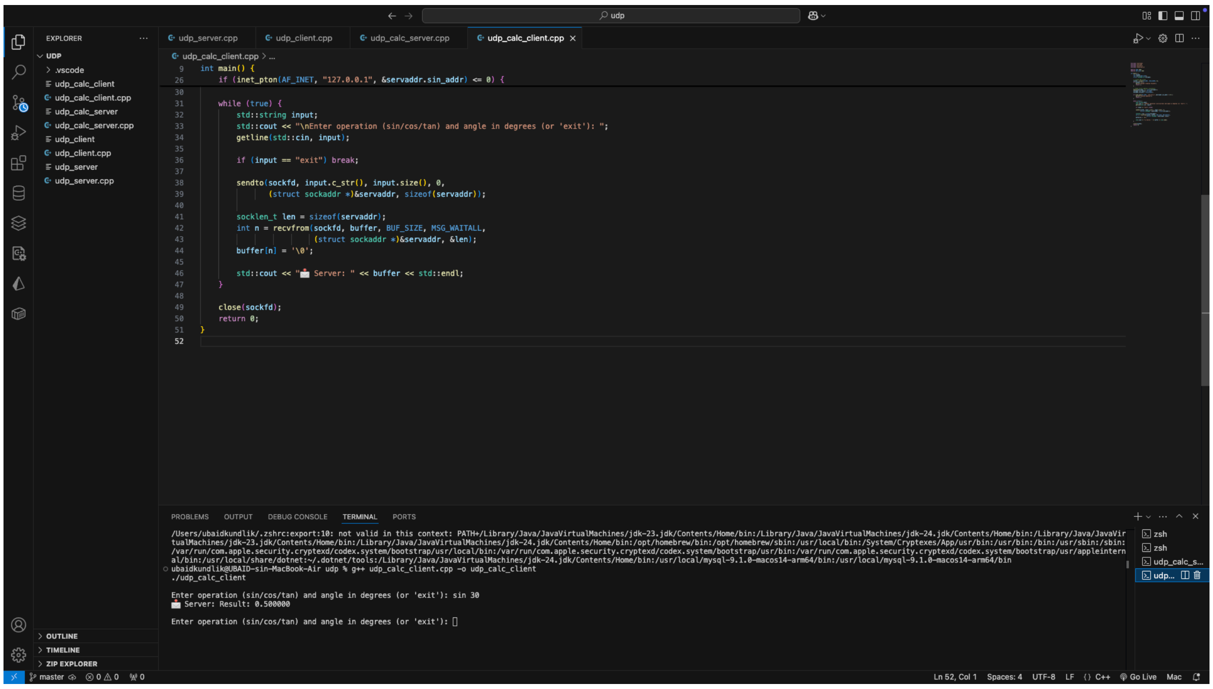
```
Enter operation (sin/cos/tan) and angle in degrees (or 'exit'):
```

1. After completion, received_sample.txt will be created on the client side.

- Client sends `"sin 30"` over UDP to server.
- Server parses it, converts `30°` → radians, computes `sin(30°) = 0.5`.
- Server sends back the result string.

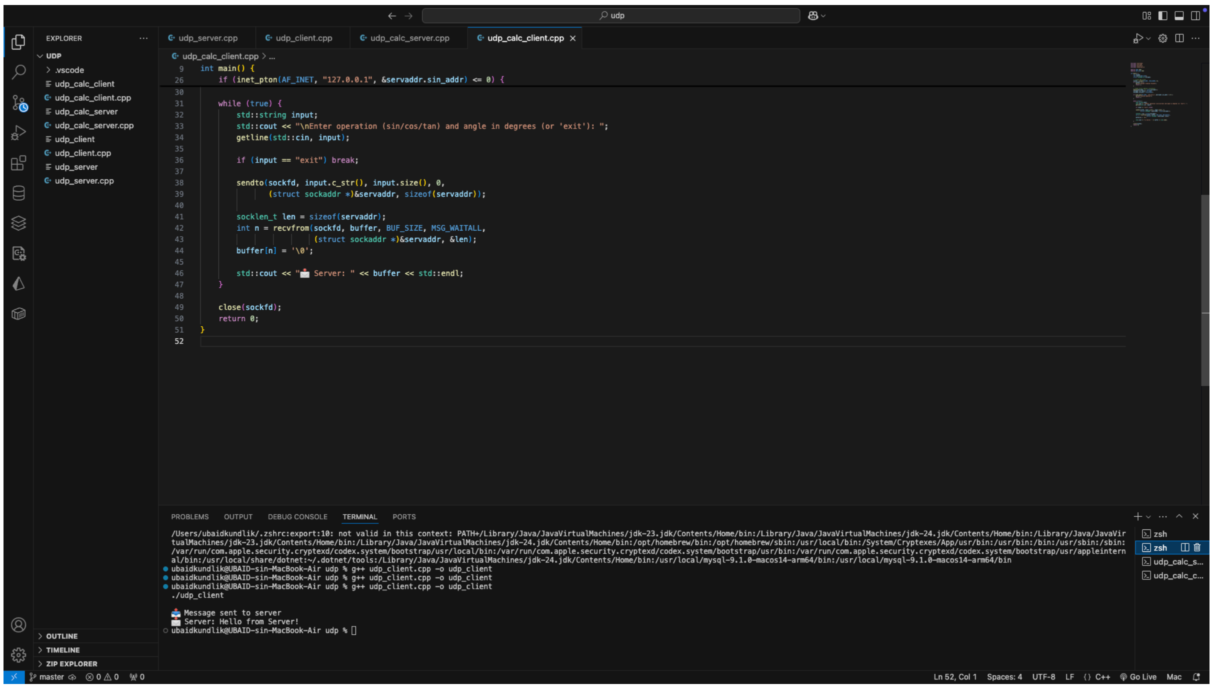
Server

OUTPUTS :



```cpp
int main() {
    if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {

    while (true) {
        std::string input;
        std::cout << "\nEnter operation (sin/cos/tan) and angle in degrees (or 'exit'): ";
        getline(std::cin, input);

        if (input == "exit") break;

        sendto(sockfd, input.c_str(), input.size(), 0,
               (struct sockaddr *)&servaddr, sizeof(servaddr));

        socklen_t len = sizeof(servaddr);
        int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                         (struct sockaddr *)&servaddr, &len);
        buffer[n] = '\0';

        std::cout << "📩 Server: " << buffer << std::endl;
    }

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

```
/Users/ubaidkundlik/.zshrc:export:10: not valid in this context: PATH+/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/Library/Java/JavaVir
tualMachines/jdk-23.jdk/Contents/Home/bin:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:
/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleintern
al/bin:/usr/local/share/dotnet:~/.dotnet/tools:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/usr/local/mysql-9.1.0-macos14-arm64/bin:/usr/local/mysql-9.1.0-macos14-arm64/bin
ubaidkundlik@UBAID-sin-MacBook-Air udp % g++ udp_calc_client.cpp -o udp_calc_client
./udp_calc_client

Enter operation (sin/cos/tan) and angle in degrees (or 'exit'): sin 30
📩 Server: Result: 0.500000

Enter operation (sin/cos/tan) and angle in degrees (or 'exit'): 
```



```cpp
int main() {
    if (inet_pton(AF_INET, "127.0.0.1", &servaddr.sin_addr) <= 0) {

    while (true) {
        std::string input;
        std::cout << "\nEnter operation (sin/cos/tan) and angle in degrees (or 'exit'): ";
        getline(std::cin, input);

        if (input == "exit") break;

        sendto(sockfd, input.c_str(), input.size(), 0,
               (struct sockaddr *)&servaddr, sizeof(servaddr));

        socklen_t len = sizeof(servaddr);
        int n = recvfrom(sockfd, buffer, BUF_SIZE, MSG_WAITALL,
                         (struct sockaddr *)&servaddr, &len);
        buffer[n] = '\0';

        std::cout << "📩 Server: " << buffer << std::endl;
    }

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

```
/Users/ubaidkundlik/.zshrc:export:10: not valid in this context: PATH+/Library/Java/JavaVirtualMachines/jdk-23.jdk/Contents/Home/bin:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/Library/Java/JavaVir
tualMachines/jdk-23.jdk/Contents/Home/bin:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:
/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleintern
al/bin:/usr/local/share/dotnet:~/.dotnet/tools:/Library/Java/JavaVirtualMachines/jdk-24.jdk/Contents/Home/bin:/usr/local/mysql-9.1.0-macos14-arm64/bin:/usr/local/mysql-9.1.0-macos14-arm64/bin
ubaidkundlik@UBAID-sin-MacBook-Air udp % g++ udp_client.cpp -o udp_client
ubaidkundlik@UBAID-sin-MacBook-Air udp % g++ udp_client.cpp -o udp_client
ubaidkundlik@UBAID-sin-MacBook-Air udp % g++ udp_client.cpp -o udp_client
./udp_client

📤 Message sent to server
📩 Server: Hello from Server!
ubaidkundlik@UBAID-sin-MacBook-Air udp % 
```

5. Conclusion

- This assignment successfully demonstrates client–server communication using UDP sockets for both simple message exchange and trigonometric calculations. It proves that UDP can provide fast, connectionless communication suitable for lightweight network applications.