Bansilal Ramnath Agarwal Charitable Trust's
VISHWAKARMA INSTITUTE OF TECHNOLOGY – PUNE

COMPUTER NETWORKS

Branch	CSE - AIML
Division	A
Batch	2
GR-no	12311305
Roll no	04
Name	Adwyte Karandikar

Experiment No. 3b:

TITLE:
Client-Server Programming using UDP Berkeley Socket Primitives.

PROBLEM STATEMENT:

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)

THEORY:

This experiment focuses on building a client-server application using the User Datagram Protocol (UDP) via the Berkeley Sockets API.

1. Berkeley Sockets API

The Berkeley Sockets API is a standard application programming interface (API) for network communication. It provides a set of functions and data structures that allow programs to create and manage network connections. A socket acts as an endpoint for sending or receiving data across a computer network.

2. User Datagram Protocol (UDP)

UDP is one of the core protocols of the Internet Protocol (IP) suite. Its key characteristics are:

- Connectionless: Unlike TCP, UDP does not establish a dedicated connection before sending data. Each packet (or datagram) is sent independently and is routed separately.
- Unreliable: UDP does not guarantee that datagrams will be delivered, arrive in order, or be free of duplicates. There is no acknowledgment, retransmission, or flow control.
- Lightweight: The lack of connection management and reliability checks makes UDP very fast and efficient, with low overhead. It is ideal for applications where speed is more critical than reliability, such as video streaming, online gaming, and DNS lookups.

3. Client-Server Model

This is a distributed application structure that partitions tasks between providers of a resource or service, called servers, and service requesters, called clients.

- **Server:** A program that waits for and fulfills requests from clients. In our case, the server binds to a specific port and listens for incoming datagrams, processes the data, and sends a response.
- **Client:** A program that initiates communication by sending a request to a server. It needs to know the server's IP address and port number to send its datagram.

4. Key Socket Functions for UDP

The following Berkeley socket primitives are used in this experiment:

- socket(domain, type, protocol): Creates a new socket.
    - i. domain: AF_INET specifies the IPv4 protocol family.
    - ii. type: SOCK_DGRAM specifies that this is a UDP socket (datagram-based).

- bind(sockfd, &addr, addrlen): (Server-side) Associates the socket (sockfd) with a specific network address and port number contained in the sockaddr struct. This allows clients to find and send messages to the server.

- sendto(sockfd, buffer, len, flags, &dest_addr, addrlen): Sends data from buffer through the socket to a specific destination address dest_addr.

- recvfrom(sockfd, buffer, len, flags, &src_addr, &addrlen): Receives data into buffer from the socket. It also populates src_addr with the address of the client that sent the data, which is crucial for the server to know where to send its reply.

- close(sockfd): Closes the socket and releases the system resources.

PROCEDURE:

The following steps outline how to compile and run the client-server programs in a Linux/Unix environment. You will need two separate terminal windows: one for the server and one for the client.

Part a: Hello to Each other

Compilation:
Open a terminal and navigate to the directory where you saved the files udp_hello_server.cpp and udp_hello_client.cpp. Compile the programs using the g++ compiler:

```
# Compile the server program
g++ udp_hello_server.cpp -o hello_server

# Compile the client program
g++ udp_hello_client.cpp -o hello_client
```

Execution:

- **Terminal 1 (Server):** Run the compiled server executable. It will wait for a message from a client.

```
./hello_server
```

- **Terminal 2 (Client):** Run the compiled client executable, providing the server's IP address and port number as command-line arguments. If you are running both on the same machine, use 127.0.0.1 as the IP. The port is 9000 as defined in the server code.

```
./hello_client 127.0.0.1 9000
```

Part b: Calculator (Trigonometry)

Compilation:
Open a terminal and navigate to the directory where you saved udp_calc_server.cpp and udp_calc_client.cpp. Compile the programs, making sure to link the math library (-lm) for the server code.

```
# Compile the server program (with math library)
g++ udp_calc_server.cpp -o calc_server -lm

# Compile the client program
g++ udp_calc_client.cpp -o calc_client
```

Execution:

- **Terminal 1 (Server):** Run the compiled calculator server. It will start and wait for expressions from the client.

```
./calc_server
```

- **Terminal 2 (Client):** Run the compiled calculator client, providing the server's IP (127.0.0.1 for local) and port (9001).

```
./calc_client 127.0.0.1 9001
```

- You can now enter trigonometric expressions in the client terminal (e.g., sin 30 deg, cos 1.57 rad). Type quit to exit the client and server.

A. Hello message exchange

1. udp_hello_sever.cpp

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

#define PORT 9000
#define BUF_SIZE 1024

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

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

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

    if (bind(sockfd, (const struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    socklen_t len = sizeof(cliaddr);
    int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr *)&cliaddr,
&len);
    buffer[n] = '\0';
    std::cout << "Client: " << buffer << std::endl;

    const char *hello = "Hello from server";
    sendto(sockfd, hello, strlen(hello), 0, (const struct sockaddr *)&cliaddr,
len);

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

2. udp_hello_client.cpp

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

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <server_ip> <port>" << std::endl;
        return 1;
    }

    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(atoi(argv[2]));
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);

    const char *hello = "Hello from client";
    sendto(sockfd, hello, strlen(hello), 0, (const struct sockaddr *)&servaddr,
sizeof(servaddr));
```

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

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

B. Calculator (Trigonometry)

1) udp_calc_sever.cpp

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

#define PORT 9001
#define BUF_SIZE 1024

double toRadians(double value, bool isDeg) {
    return isDeg ? value * M_PI / 180.0 : value;
}

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

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    if (bind(sockfd, (struct sockaddr *)&servaddr, sizeof(servaddr)) < 0) {
        perror("Bind failed");
        exit(EXIT_FAILURE);
    }

    socklen_t len = sizeof(cliaddr);
    while (true) {
        int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr
*)&cliaddr, &len);
        buffer[n] = '\0';
        std::string input(buffer);
        if (input == "quit") break;

        char func[10], unit[10] = "rad";
        double value;
        sscanf(buffer, "%s %lf %s", func, &value, unit);

        bool isDeg = (strcmp(unit, "deg") == 0);
        double result = 0.0;

        if (strcmp(func, "sin") == 0)
            result = sin(toRadians(value, isDeg));
        else if (strcmp(func, "cos") == 0)
            result = cos(toRadians(value, isDeg));
        else if (strcmp(func, "tan") == 0)
            result = tan(toRadians(value, isDeg));
        else
            snprintf(buffer, BUF_SIZE, "Invalid function");

        snprintf(buffer, BUF_SIZE, "Result: %.6f", result);
        sendto(sockfd, buffer, strlen(buffer), 0, (struct sockaddr *)&cliaddr,
len);
    }

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

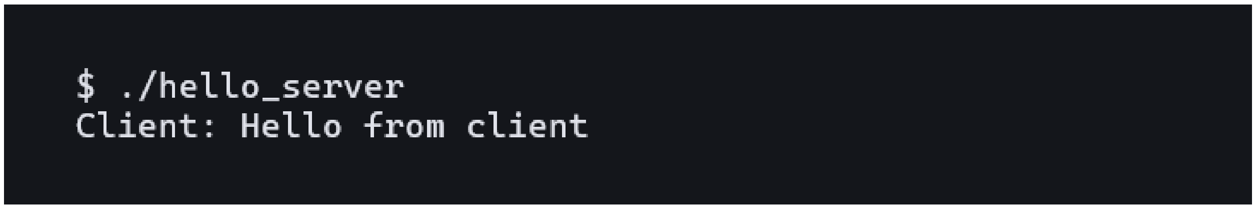
2) udp_calc_client.cpp

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

#define BUF_SIZE 1024

int main(int argc, char *argv[]) {
    if (argc != 3) {
        std::cerr << "Usage: " << argv[0] << " <server_ip> <port>" << std::endl;
        return 1;
    }

    int sockfd;
    char buffer[BUF_SIZE];
    struct sockaddr_in servaddr;

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd < 0) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(atoi(argv[2]));
    servaddr.sin_addr.s_addr = inet_addr(argv[1]);

    socklen_t len = sizeof(servaddr);

    while (true) {
        std::cout << "Enter expression (e.g., sin 30 deg) or quit: ";
        std::string expr;
        std::getline(std::cin, expr);

        sendto(sockfd, expr.c_str(), expr.length(), 0, (struct sockaddr
*)&servaddr, len);

        if (expr == "quit") break;

        int n = recvfrom(sockfd, buffer, BUF_SIZE, 0, (struct sockaddr
*)&servaddr, &len);
        buffer[n] = '\0';
        std::cout << buffer << std::endl;
    }

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

SCREENSHOTS/OUTPUT:

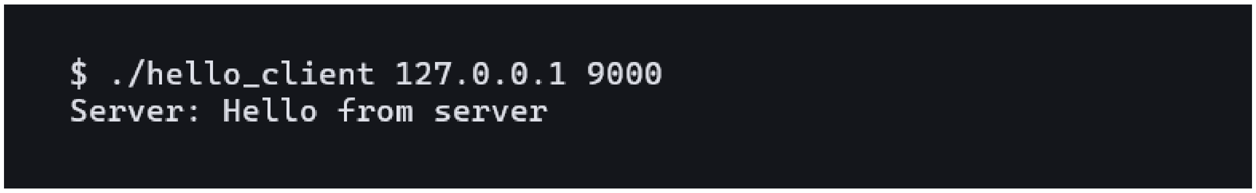
A. **Hello Program Output**

i. **Terminal 1: Server Output**



```
$ ./hello_server
Client: Hello from client
```

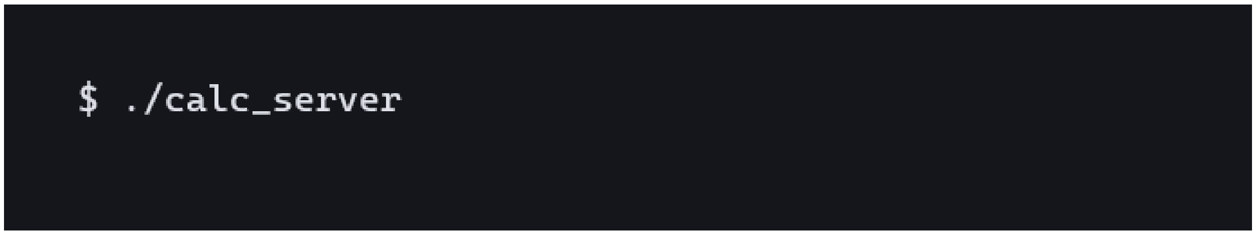
ii. **Terminal 2: Client Output**



```
$ ./hello_client 127.0.0.1 9000
Server: Hello from server
```

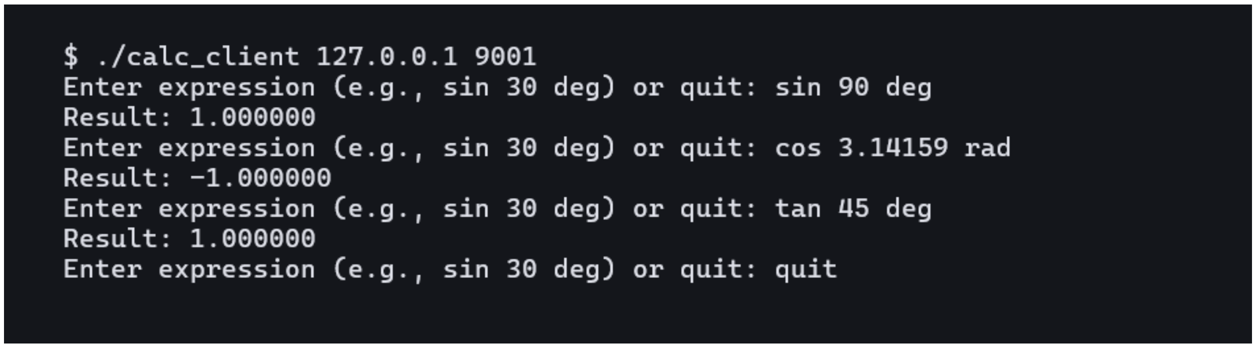
B. **Calculator Program Output**

iii. **Terminal 1: Server Output**



```
$ ./calc_server
```

iv. **Terminal 2: Client Output**



```
$ ./calc_client 127.0.0.1 9001
Enter expression (e.g., sin 30 deg) or quit: sin 90 deg
Result: 1.000000
Enter expression (e.g., sin 30 deg) or quit: cos 3.14159 rad
Result: -1.000000
Enter expression (e.g., sin 30 deg) or quit: tan 45 deg
Result: 1.000000
Enter expression (e.g., sin 30 deg) or quit: quit
```

CONCLUSION:

This experiment successfully demonstrated the implementation of a client-server model using the User Datagram Protocol (UDP) with Berkeley socket primitives. Key takeaways from this assignment include:

1. **UDP as a Connectionless Protocol:** Unlike TCP, UDP does not establish a persistent connection. Messages (datagrams) are sent independently, which was observed in how sendto() and recvfrom() were used to transmit data in single, discrete packets.

2. **Socket Programming Fundamentals:** We practiced the core steps of socket programming:
    - socket(): Creating an endpoint for communication.
    - bind(): Assigning a specific port and address to the server socket so clients know where to send messages.
    - sendto() & recvfrom(): The primary functions for transmitting and receiving data over UDP, which require specifying the destination/source address with each call.
    - close(): Releasing the socket resource.

3. **Client vs. Server Roles:** The distinction between the server and client was clear. The server binds to a port and waits passively, while the client actively initiates communication by sending a message to the server's known address and port.

4. Practical Application: Two applications were developed—a simple "Hello" exchange and a more functional trigonometric calculator. This showed how the same underlying UDP framework can be used to build different network services by varying the data being exchanged and the logic for processing it on the server side.

Overall, the assignment provided hands-on experience in building network applications and a practical understanding of the stateless, connectionless nature of UDP communication.