**Bansilal Ramnath Agarwal Charitable Trust’s**

Vishwakarma Institute of Technology, Pune-37

*(Anautonomous Institute of Savitribai Phule Pune University)*

**Department of Computer Engineering**

|  |  |
| --- | --- |
| **Division** | **A** |
| **Batch** | **B2** |
| **GR-no** | **12110353** |
| **Roll no** | **10** |
| **Name** | **Aneesh Hemant Dighe** |

For a chat application using UDP Berkeley socket primitives, you can implement both peer-to-peer and client-server modes. UDP (User Datagram Protocol) is a connectionless, lightweight protocol suitable for real-time communication like chats due to its low overhead and speed. Here's a high-level overview for both modes:

Client-Server Mode:

Server Setup: Implement a server that listens on a specific UDP port using socket() and bind().

Client Connection: Clients create sockets with socket() and send messages to the server's IP address and port using sendto().

Server Processing: The server receives messages using recvfrom(), processes them, and potentially broadcasts them to connected clients.

Peer-to-Peer Mode:

Client Setup: Each peer has a client application that creates a UDP socket.

Peer Discovery: Peers can broadcast their presence on the network by sending a discovery message.

Peer-to-Peer Communication: When a peer wants to chat with another, they establish a direct UDP connection by sending messages directly to the other peer's IP and port.

Handling NAT: Consider implementing techniques like NAT traversal (e.g., STUN or TURN) for cases where peers are behind NAT devices.

Key considerations for both modes:

Error handling: Handle potential packet loss and out-of-order delivery, as UDP doesn't guarantee reliable communication.

Message encoding: Define a message format to structure chat data.

Security: Consider encryption and authentication for secure communication.

Scalability: Ensure the application can handle multiple clients or peer connections simultaneously.

Real-time updates: Implement mechanisms to update the chat interface in real-time when new messages arrive.

***CODE:***

client.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define SERVER\_IP "127.0.0.1"

#define PORT 12345

#define BUFSIZE 1024

void error(const char \*msg) {

    perror(msg);

    exit(1);

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        error("Error initializing Winsock");

    }

    SOCKET clientSocket;

    struct sockaddr\_in serverAddr;

    char buffer[BUFSIZE];

    // Create socket

    clientSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (clientSocket == INVALID\_SOCKET)

        error("Error opening socket");

    // Initialize server address structure

    serverAddr.sin\_family = AF\_INET;

    serverAddr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    serverAddr.sin\_port = htons(PORT);

    printf("Enter messages to send (type 'exit' to quit):\n");

    while (1) {

        // Read user input

        printf("You: ");

        fgets(buffer, BUFSIZE, stdin);

        buffer[strcspn(buffer, "\n")] = '\0'; // Remove trailing newline

        // Send the message to the server

        if (sendto(clientSocket, buffer, strlen(buffer), 0, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)

            error("Error sending data to server");

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

            printf("Exiting...\n");

            break;

        }

        // Receive the echoed message from the server

        int bytesRead = recvfrom(clientSocket, buffer, BUFSIZE, 0, NULL, NULL);

        if (bytesRead == SOCKET\_ERROR)

            error("Error receiving data from server");

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

        printf("Server: %s\n", buffer);

    }

    closesocket(clientSocket);

    WSACleanup();

    return 0;

}

***Server.c***

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <winsock2.h>

#pragma comment(lib, "ws2\_32.lib")

#define PORT 12345

#define BUFSIZE 1024

void error(const char \*msg) {

    perror(msg);

    exit(1);

}

int main() {

    WSADATA wsaData;

    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {

        error("Error initializing Winsock");

    }

    SOCKET serverSocket;

    struct sockaddr\_in serverAddr, clientAddr;

    int clientAddrLen = sizeof(clientAddr);

    char buffer[BUFSIZE];

    // Create socket

    serverSocket = socket(AF\_INET, SOCK\_DGRAM, 0);

    if (serverSocket == INVALID\_SOCKET)

        error("Error opening socket");

    // Initialize server address structure

    serverAddr.sin\_family = AF\_INET;

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

    serverAddr.sin\_port = htons(PORT);

    // Bind the socket

    if (bind(serverSocket, (struct sockaddr \*)&serverAddr, sizeof(serverAddr)) == SOCKET\_ERROR)

        error("Error binding socket");

    printf("Server listening on port %d...\n", PORT);

    while (1) {

        int bytesRead = recvfrom(serverSocket, buffer, BUFSIZE, 0, (struct sockaddr \*)&clientAddr, &clientAddrLen);

        if (bytesRead == SOCKET\_ERROR)

            error("Error receiving data");

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

        printf("Client (%s:%d): %s\n", inet\_ntoa(clientAddr.sin\_addr), ntohs(clientAddr.sin\_port), buffer);

        // Echo the message back to the client

        if (sendto(serverSocket, buffer, bytesRead, 0, (struct sockaddr \*)&clientAddr, sizeof(clientAddr)) == SOCKET\_ERROR)

            error("Error sending data back to client");

    }

    closesocket(serverSocket);

    WSACleanup();

    return 0;

}

***OUTPUT:***

PS D:\5th Sem\CN(Computer Networks)\Assignments\UDP> ./client.exe

Enter messages to send (type 'exit' to quit):

You: hello

Server: hello

You: aneesh

Server: aneesh

You: dighe

Server: dighe

You: exit

Exiting...

PS D:\5th Sem\CN(Computer Networks)\Assignments\UDP>

./server.exe

Server listening on port 12345...

Client (127.0.0.1:50874): hello

Client (127.0.0.1:50874): aneesh

Client (127.0.0.1:50874): dighe

Client (127.0.0.1:50874): exit