**Tcp socket**

SERVER

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int server\_socket, client\_socket;  
    struct sockaddr\_in server\_addr, client\_addr;  
    socklen\_t client\_addr\_len = sizeof(client\_addr);  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
    if (server\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    // Bind the socket to the server address  
    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error binding socket");  
        close(server\_socket);  
        exit(1);  
    }

    // Listen for incoming connections  
    if (listen(server\_socket, 5) == -1) {  
        perror("Error listening for connections");  
        close(server\_socket);  
        exit(1);  
    }

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

    // Accept a connection from a client  
    client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);  
    if (client\_socket == -1) {  
        perror("Error accepting connection");  
        close(server\_socket);  
        exit(1);  
    }

    printf("Client connected.\n");

    // Receive and echo data  
    while (1) {  
        int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);  
        if (bytes\_received <= 0) {  
            printf("Connection closed by client.\n");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Received: %s", buffer);

        // Echo the received data back to the client  
        send(client\_socket, buffer, strlen(buffer), 0);  
    }

    // Close sockets  
    close(client\_socket);  
    close(server\_socket);

    return 0;  
}

CLIENT

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address  
#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int client\_socket;  
    struct sockaddr\_in server\_addr;  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
    if (client\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Connect to the server  
    if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error connecting to server");  
        close(client\_socket);  
        exit(1);  
    }

    printf("Connected to server at %s:%d\n", SERVER\_IP, PORT);

    // Send and receive data  
    while (1) {  
        printf("Enter a message to send (or 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }

        // Send the message to the server  
        send(client\_socket, buffer, strlen(buffer), 0);

        // Receive and print the echo from the server  
        int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);  
        if (bytes\_received <= 0) {  
            printf("Connection closed by server.\n");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Received: %s", buffer);  
    }

    // Close the socket  
    close(client\_socket);

    return 0;  
}

**Udp socket**

**SERVER:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int server\_socket;  
    struct sockaddr\_in server\_addr, client\_addr;  
    socklen\_t client\_addr\_len = sizeof(client\_addr);  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);  
    if (server\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    // Bind the socket to the server address  
    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error binding socket");  
        close(server\_socket);  
        exit(1);  
    }

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

    // Receive and echo data  
    while (1) {  
        int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &client\_addr\_len);  
        if (bytes\_received <= 0) {  
            perror("Error receiving data");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Received: %s", buffer);

        // Echo the received data back to the client  
        sendto(server\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&client\_addr, client\_addr\_len);  
    }

    // Close the socket  
    close(server\_socket);

    return 0;  
}

**CLIENT:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address  
#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int client\_socket;  
    struct sockaddr\_in server\_addr;  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);  
    if (client\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Send and receive data  
    while (1) {  
        printf("Enter a message to send (or 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }

        // Send the message to the server  
        sendto(client\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

        // Receive and print the echo from the server  
        int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, NULL, NULL);  
        if (bytes\_received <= 0) {  
            perror("Error receiving data");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Received: %s", buffer);  
    }

    // Close the socket  
    close(client\_socket);

    return 0;  
}

**tcp chat**

**SERVER:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int server\_socket, client\_socket;  
    struct sockaddr\_in server\_addr, client\_addr;  
    socklen\_t client\_addr\_len = sizeof(client\_addr);  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
    if (server\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    // Bind the socket to the server address  
    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error binding socket");  
        close(server\_socket);  
        exit(1);  
    }

    // Listen for incoming connections  
    if (listen(server\_socket, 5) == -1) {  
        perror("Error listening for connections");  
        close(server\_socket);  
        exit(1);  
    }

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

    // Accept a connection from a client  
    client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_addr\_len);  
    if (client\_socket == -1) {  
        perror("Error accepting connection");  
        close(server\_socket);  
        exit(1);  
    }

    printf("Client connected.\n");

    // Chat loop  
    while (1) {  
        // Receive a message from the client  
        int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);  
        if (bytes\_received <= 0) {  
            printf("Connection closed by client.\n");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Client: %s", buffer);

        // Prompt for a reply  
        printf("Server (Type 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        // Send the reply to the client  
        send(client\_socket, buffer, strlen(buffer), 0);

        // Check if the server wants to quit  
        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }  
    }

    // Close sockets  
    close(client\_socket);  
    close(server\_socket);

    return 0;  
}

**CLIENT:**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address  
#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int client\_socket;  
    struct sockaddr\_in server\_addr;  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    client\_socket = socket(AF\_INET, SOCK\_STREAM, 0);  
    if (client\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Connect to the server  
    if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error connecting to server");  
        close(client\_socket);  
        exit(1);  
    }

    printf("Connected to server at %s:%d\n", SERVER\_IP, PORT);

    // Chat loop  
    while (1) {  
        // Prompt for a message to send  
        printf("Client (Type 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        // Send the message to the server  
        send(client\_socket, buffer, strlen(buffer), 0);

        // Check if the client wants to quit  
        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }

        // Receive a message from the server  
        int bytes\_received = recv(client\_socket, buffer, sizeof(buffer), 0);  
        if (bytes\_received <= 0) {  
            printf("Connection closed by server.\n");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Server: %s", buffer);  
    }

    // Close the socket  
    close(client\_socket);

    return 0;  
}

**udp chat**

**SERVER**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int server\_socket;  
    struct sockaddr\_in server\_addr, client\_addr;  
    socklen\_t client\_addr\_len = sizeof(client\_addr);  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    server\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);  
    if (server\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

    // Bind the socket to the server address  
    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {  
        perror("Error binding socket");  
        close(server\_socket);  
        exit(1);  
    }

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

    // Receive and send data  
    while (1) {  
        int bytes\_received = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &client\_addr\_len);  
        if (bytes\_received <= 0) {  
            perror("Error receiving data");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Client: %s", buffer);

        // Prompt for a reply  
        printf("Server (Type 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        // Send the reply to the client  
        sendto(server\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&client\_addr, client\_addr\_len);

        // Check if the server wants to quit  
        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }  
    }

    // Close the socket  
    close(server\_socket);

    return 0;  
}

**CLIENT**

#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
#include <unistd.h>  
#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1" // Change this to the server's IP address  
#define PORT 12345  
#define BUFFER\_SIZE 1024

int main() {  
    int client\_socket;  
    struct sockaddr\_in server\_addr;  
    char buffer[BUFFER\_SIZE];

    // Create a socket  
    client\_socket = socket(AF\_INET, SOCK\_DGRAM, 0);  
    if (client\_socket == -1) {  
        perror("Error creating socket");  
        exit(1);  
    }

    // Set up the server address struct  
    server\_addr.sin\_family = AF\_INET;  
    server\_addr.sin\_port = htons(PORT);  
    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Send and receive data  
    while (1) {  
        // Prompt for a message to send  
        printf("Client (Type 'quit' to exit): ");  
        fgets(buffer, sizeof(buffer), stdin);

        // Send the message to the server  
        sendto(client\_socket, buffer, strlen(buffer), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr));

        // Check if the client wants to quit  
        if (strcmp(buffer, "quit\n") == 0) {  
            break;  
        }

        // Receive and print the server's reply  
        int bytes\_received = recvfrom(client\_socket, buffer, sizeof(buffer), 0, NULL, NULL);  
        if (bytes\_received <= 0) {  
            perror("Error receiving data");  
            break;  
        }  
        buffer[bytes\_received] = '\0';  
        printf("Server: %s", buffer);  
    }

    // Close the socket  
    close(client\_socket);

    return 0;  
}

**Arp and rarp**

#include <stdio.h>

#include <string.h>

struct ARPEntry {

char ip[20];

char mac[20];

};

void displayTable(struct ARPEntry table[], int size) {

printf("\nIP Address\tMAC Address\n");

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

for (int i = 0; i < size; i++) {

printf("%s\t%s\n", table[i].ip, table[i].mac);

}

}

void simulateARP(struct ARPEntry table[], int size, char ip[]) {

for (int i = 0; i < size; i++) {

if (strcmp(ip, table[i].ip) == 0) {

printf("\nARP Response: MAC Address for IP %s is %s\n", ip, table[i].mac);

return;

}

}

printf("\nARP Response: No MAC Address found for IP %s\n", ip);

}

void simulateRARP(struct ARPEntry table[], int size, char mac[]) {

for (int i = 0; i < size; i++) {

if (strcmp(mac, table[i].mac) == 0) {

printf("\nRARP Response: IP Address for MAC %s is %s\n", mac, table[i].ip);

return;

}

}

printf("\nRARP Response: No IP Address found for MAC %s\n", mac);

}

int main() {

struct ARPEntry table[5] = {

{"192.168.0.1", "00:0a:95:9d:68:16"},

{"192.168.0.2", "00:0a:95:9d:68:17"},

{"192.168.0.3", "00:0a:95:9d:68:18"},

{"192.168.0.4", "00:0a:95:9d:68:19"},

{"192.168.0.5", "00:0a:95:9d:68:20"}

};

char choice;

char ip[20], mac[20];

displayTable(table, 5);

printf("\nChoose Simulation:\n1. ARP\n2. RARP\n");

scanf(" %c", &choice);

switch (choice) {

case '1':

printf("\nEnter IP address to simulate ARP: ");

scanf("%s", ip);

simulateARP(table, 5, ip);

break;

case '2':

printf("\nEnter MAC address to simulate RARP: ");

scanf("%s", mac);

simulateRARP(table, 5, mac);

break;

default:

printf("\nInvalid Choice\n");

}

return 0;

}