**Tcp Server:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port 8080

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

while (1) {

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

int valread = read(new\_socket, buffer, BUFFER\_SIZE);

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

send(new\_socket, buffer, valread, 0);

printf("Echoed message back to client\n");

close(new\_socket);

}

close(server\_fd);

return 0;

}

**TCP client:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

printf("\n Socket creation error \n");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

printf("\nInvalid address/ Address not supported \n");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

printf("\nConnection Failed \n");

return -1;

}

printf("Enter a message: ");

fgets(buffer, BUFFER\_SIZE, stdin);

send(sock, buffer, strlen(buffer), 0);

printf("Message sent\n");

int valread = read(sock, buffer, BUFFER\_SIZE);

printf("Echoed message from server: %s\n", buffer);

close(sock);

return 0;

}

**UDP server:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in servaddr, cliaddr;

socklen\_t len;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

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

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

// Filling server information

servaddr.sin\_family = AF\_INET; // IPv4

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

servaddr.sin\_port = htons(PORT);

// Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

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

while (1) {

len = sizeof(cliaddr); // len is value/result

int n = recvfrom(sockfd, (char \*)buffer, BUFFER\_SIZE, MSG\_WAITALL, (struct sockaddr \*)&cliaddr, &len);

buffer[n] = '\0';

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

const char \*response = "Hello, Client!";

sendto(sockfd, response, strlen(response), MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr, len);

printf("Response sent to client\n");

}

close(sockfd);

return 0;

}

**UDP Client:**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

char buffer[BUFFER\_SIZE];

struct sockaddr\_in servaddr;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

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

// Filling server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

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

const char \*message = "Hello, Server!";

sendto(sockfd, message, strlen(message), MSG\_CONFIRM, (const struct sockaddr \*)&servaddr, sizeof(servaddr));

printf("Message sent to server\n");

int n = recvfrom(sockfd, (char \*)buffer, BUFFER\_SIZE, MSG\_WAITALL, NULL, NULL);

buffer[n] = '\0';

printf("Response from server: %s\n", buffer);

close(sockfd);

return 0;

}

**Example 3:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, new\_socket;

struct sockaddr\_in address;

int addrlen = sizeof(address);

char buffer[BUFFER\_SIZE] = {0};

char file\_name[BUFFER\_SIZE] = {0};

FILE \*file;

// Creating socket file descriptor

if ((server\_fd = socket(AF\_INET, SOCK\_STREAM, 0)) == 0) {

perror("Socket failed");

exit(EXIT\_FAILURE);

}

// Forcefully attaching socket to the port

address.sin\_family = AF\_INET;

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

address.sin\_port = htons(PORT);

if (bind(server\_fd, (struct sockaddr \*)&address, sizeof(address)) < 0) {

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

if (listen(server\_fd, 3) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

while (1) {

if ((new\_socket = accept(server\_fd, (struct sockaddr \*)&address, (socklen\_t\*)&addrlen)) < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Receive the file name

read(new\_socket, file\_name, BUFFER\_SIZE);

printf("Receiving file: %s\n", file\_name);

// Open file for writing

file = fopen(file\_name, "wb");

if (file == NULL) {

perror("File open failed");

close(new\_socket);

continue;

}

// Receive the file contents

int bytes\_read;

while ((bytes\_read = read(new\_socket, buffer, BUFFER\_SIZE)) > 0) {

fwrite(buffer, sizeof(char), bytes\_read, file);

}

printf("File received and saved as %s\n", file\_name);

fclose(file);

close(new\_socket);

}

close(server\_fd);

return 0;

}

**TCP Client:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sock = 0;

struct sockaddr\_in serv\_addr;

char buffer[BUFFER\_SIZE] = {0};

char file\_path[BUFFER\_SIZE];

FILE \*file;

// Creating socket file descriptor

if ((sock = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) {

perror("Socket creation error");

return -1;

}

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_port = htons(PORT);

// Convert IPv4 and IPv6 addresses from text to binary form

if (inet\_pton(AF\_INET, "127.0.0.1", &serv\_addr.sin\_addr) <= 0) {

perror("Invalid address / Address not supported");

return -1;

}

if (connect(sock, (struct sockaddr \*)&serv\_addr, sizeof(serv\_addr)) < 0) {

perror("Connection failed");

return -1;

}

// Prompt user for file path

printf("Enter the file path to send: ");

scanf("%s", file\_path);

// Open the file for reading

file = fopen(file\_path, "rb");

if (file == NULL) {

perror("File open failed");

return -1;

}

// Send the file name

char \*file\_name = strrchr(file\_path, '/');

if (file\_name) {

file\_name++; // Get the actual file name

} else {

file\_name = file\_path; // In case of no path, just the file name

}

send(sock, file\_name, strlen(file\_name), 0);

// Send the file contents

int bytes\_read;

while ((bytes\_read = fread(buffer, sizeof(char), BUFFER\_SIZE, file)) > 0) {

send(sock, buffer, bytes\_read, 0);

}

printf("File sent successfully\n");

fclose(file);

close(sock);

return 0;

}

**Example 4:**

Broadcast:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BROADCAST\_IP "255.255.255.255"

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in broadcast\_addr;

int broadcast\_enable = 1;

char message[BUFFER\_SIZE];

// Create socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set socket options to allow broadcast

if (setsockopt(sockfd, SOL\_SOCKET, SO\_BROADCAST, &broadcast\_enable, sizeof(broadcast\_enable)) < 0) {

perror("Error in setting broadcast option");

close(sockfd);

exit(EXIT\_FAILURE);

}

// Set up the broadcast address struct

memset(&broadcast\_addr, 0, sizeof(broadcast\_addr));

broadcast\_addr.sin\_family = AF\_INET;

broadcast\_addr.sin\_port = htons(PORT);

broadcast\_addr.sin\_addr.s\_addr = inet\_addr(BROADCAST\_IP);

while (1) {

printf("Enter message to broadcast (or 'exit' to quit): ");

fgets(message, BUFFER\_SIZE, stdin);

// Exit if user inputs 'exit'

if (strncmp(message, "exit", 4) == 0) {

break;

}

// Broadcast the message

if (sendto(sockfd, message, strlen(message), 0, (struct sockaddr \*)&broadcast\_addr, sizeof(broadcast\_addr)) < 0) {

perror("Broadcast message failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

printf("Broadcast message sent: %s", message);

}

close(sockfd);

return 0;

}

**Broadcast Client:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 8080

#define BUFFER\_SIZE 1024

int main() {

int sockfd;

struct sockaddr\_in recv\_addr;

char buffer[BUFFER\_SIZE];

socklen\_t addr\_len = sizeof(recv\_addr);

// Create socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Set up the receive address struct

memset(&recv\_addr, 0, sizeof(recv\_addr));

recv\_addr.sin\_family = AF\_INET;

recv\_addr.sin\_port = htons(PORT);

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

// Bind the socket to the receive address and port

if (bind(sockfd, (const struct sockaddr \*)&recv\_addr, sizeof(recv\_addr)) < 0) {

perror("Bind failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

printf("Listening for broadcast messages on port %d...\n", PORT);

while (1) {

// Receive broadcast message

int n = recvfrom(sockfd, buffer, BUFFER\_SIZE, 0, (struct sockaddr \*)&recv\_addr, &addr\_len);

if (n < 0) {

perror("Receive failed");

close(sockfd);

exit(EXIT\_FAILURE);

}

buffer[n] = '\0'; // Null-terminate the received message

printf("Received broadcast message: %s", buffer);

}

close(sockfd);

return 0;

}