**Ex:4a - Computer Network Lab**

**Name:** Athithraja R

**Reg.no:** 2022503702

**4a. TCP**

**Echo\_server.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 22000

#define BUFFER\_SIZE 1024

int main() {

int server\_fd, client\_fd;

struct sockaddr\_in server\_addr, client\_addr;

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

char buffer[BUFFER\_SIZE];

// Create socket

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

if (server\_fd < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

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

server\_addr.sin\_addr.s\_addr = INADDR\_ANY; // Accept connections from any IP

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

// Bind the socket

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

perror("Bind failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

// Listen for incoming connections

if (listen(server\_fd, 5) < 0) {

perror("Listen failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

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

// Accept a client connection

client\_fd = accept(server\_fd, (struct sockaddr\*)&client\_addr, &addr\_len);

if (client\_fd < 0) {

perror("Accept failed");

close(server\_fd);

exit(EXIT\_FAILURE);

}

while (1) {

memset(buffer, 0, BUFFER\_SIZE); // Clear the buffer

ssize\_t bytes\_read = read(client\_fd, buffer, BUFFER\_SIZE);

if (bytes\_read <= 0) {

printf("Client disconnected\n");

break;

}

printf("Received: %s", buffer); // Print received message

write(client\_fd, buffer, bytes\_read); // Echo the message back

printf(“\n”);

}

// Close sockets

close(client\_fd);

close(server\_fd);

return 0;

}

**Output:**



**Echo\_client.c:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 22000

#define BUFFER\_SIZE 1024

int main() {

int client\_fd;

struct sockaddr\_in server\_addr;

// Create socket

client\_fd = socket(AF\_INET, SOCK\_STREAM, 0);

if (client\_fd < 0) {

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Configure server address

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

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

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

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

perror("Invalid address or address not supported");

close(client\_fd);

exit(EXIT\_FAILURE);

}

// Connect to the server

if (connect(client\_fd, (struct sockaddr\*)&server\_addr, sizeof(server\_addr)) < 0) {

perror("Connection failed");

close(client\_fd);

exit(EXIT\_FAILURE);

}

printf("Connected to the server\n");

char buffer[BUFFER\_SIZE];

while (1) {

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

fgets(buffer, BUFFER\_SIZE, stdin);

// Remove newline character from the input

buffer[strcspn(buffer, "\n")] = '\0';

// Send the message to the server

write(client\_fd, buffer, strlen(buffer));

// Check if the user wants to exit

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

printf("Disconnecting from the server...\n");

break;

}

// Read the echoed message from the server

memset(buffer, 0, BUFFER\_SIZE); // Clear the buffer

ssize\_t bytes\_read = read(client\_fd, buffer, BUFFER\_SIZE);

if (bytes\_read <= 0) {

printf("Server disconnected\n");

break;

}

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

}

// Close the socket

close(client\_fd);

return 0;

}

**Output:**

