29)

TCPEcho.java

```java
import java.io.*;
import java.net.*;
import java.util.Scanner;

public class TCPEcho {

    // TCP Echo Server
    public static void startServer(int port) {
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Echo Server is running on port " +
                port + "...");

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Client connected: " +
                    clientSocket.getInetAddress());

                BufferedReader input = new BufferedReader(new
                    InputStreamReader(clientSocket.getInputStream
                    ()));
                PrintWriter output = new PrintWriter(clientSocket
                    .getOutputStream(), true);

                String receivedMessage;
                while ((receivedMessage = input.readLine()) != null)
                    {
```

Output

```
Choose mode: 1) Server  2) Client
```

TCPEcho.java

```java
                while ((receivedMessage = input.readLine()) != null)
                    {
                    System.out.println("Received: " + receivedMessage
                        );
                    output.println("Echo: " + receivedMessage); //
                        Send back the same message
                }

                clientSocket.close();
                System.out.println("Client disconnected.");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // TCP Echo Client
    public static void startClient(String serverAddress, int port) {
        try (Socket socket = new Socket(serverAddress, port);
            BufferedReader userInput = new BufferedReader(new
                InputStreamReader(System.in));
            PrintWriter output = new PrintWriter(socket
                .getOutputStream(), true);
            BufferedReader input = new BufferedReader(new
                InputStreamReader(socket.getInputStream()))) {

            System.out.println("Connected to Echo Server. Type
```

Output

```
Choose mode: 1) Server  2) Client
```

TCPEcho.java

```java
            System.out.println("Connected to Echo Server. Type messages to send (type 'exit' to quit):");

            String message;
            while (true) {
                System.out.print("You: ");
                message = userInput.readLine();
                if (message.equalsIgnoreCase("exit")) {
                    break;
                }
                output.println(message);
                System.out.println("Server: " + input.readLine()); // Read echoed response
            }

            System.out.println("Closing connection...");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Main method to choose between Server and Client
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Choose mode: 1) Server  2) Client");
        int choice = scanner.nextInt();

```

Output

```
Choose mode: 1) Server  2) Client
```

TCPEcho.java

```java
            System.out.println("Closing connection...");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    // Main method to choose between Server and Client
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Choose mode: 1) Server  2) Client");
        int choice = scanner.nextInt();

        if (choice == 1) {
            startServer(5000);
        } else if (choice == 2) {
            System.out.print("Enter Server IP (localhost for local testing): ");
            String serverAddress = scanner.next();
            startClient(serverAddress, 5000);
        } else {
            System.out.println("Invalid choice.");
        }

        scanner.close();
    }
}
```

Output

```
Choose mode: 1) Server  2) Client
```

30) server

Programiz
C Online Compiler

Program

main.c

Share

Run

Output

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/socket.h>

#define PORT 8080
#define BUFFER_SIZE 1024

void handle_client(int client_socket) {
    char buffer[BUFFER_SIZE];
    int bytes_read;

    while (1) {
        memset(buffer, 0, BUFFER_SIZE);
        bytes_read = recv(client_socket, buffer, BUFFER_SIZE, 0
            );

        if (bytes_read <= 0) {
```

```
Server is listening on port 8080...
```

```c
            printf("Client disconnected.\n");
            break;
        }

        printf("Client: %s\n", buffer);
        send(client_socket, buffer, bytes_read, 0);
    }

    close(client_socket);
}

int main() {
    int server_socket, client_socket;
    struct sockaddr_in server_addr, client_addr;
    socklen_t client_len = sizeof(client_addr);

    // Create socket
    server_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (server_socket == -1) {
        perror("Failed to create socket");
```


```
Server is listening on port 8080...
```

main.c

Share Run

```c
        perror("Failed to create socket");
        exit(EXIT_FAILURE);
    }

    // Define server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(PORT);

    // Bind the socket to the specified IP and port
    if (bind(server_socket, (struct sockaddr*)&server_addr,
        sizeof(server_addr)) < 0) {
        perror("Bind failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_socket, 3) < 0) {
        perror("Listen failed");
```

Output

```
Server is listening on port 8080...
```

main.c

Share Run

```c
        perror("Listen failed");
        close(server_socket);
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        // Accept a new connection
        client_socket = accept(server_socket, (struct sockaddr
            *)&client_addr, &client_len);
        if (client_socket < 0) {
            perror("Accept failed");
            close(server_socket);
            exit(EXIT_FAILURE);
        }

        printf("New client connected.\n");
        handle_client(client_socket);
    }
```

Output

```
Server is listening on port 8080...
```

```c
    while (1) {
        // Accept a new connection
        client_socket = accept(server_socket, (struct sockaddr
            *)&client_addr, &client_len);
        if (client_socket < 0) {
            perror("Accept failed");
            close(server_socket);
            exit(EXIT_FAILURE);
        }

        printf("New client connected.\n");
        handle_client(client_socket);
    }

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

Client response

main.c

Share Run Output

```c
#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 client_socket;
    struct sockaddr_in server_addr;  // Fixed this line
    char buffer[BUFFER_SIZE];

    // Create socket
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    if (client_socket == -1) {
        perror("Socket creation failed");
        exit(EXIT_FAILURE);
    }
```

```
Connection failed: Connection refused


=== Code Exited With Errors ===
```

main.c

```c
    }

    // Define server address
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(PORT);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");  // Connect to localhost

    // Connect to the server
    if (connect(client_socket, (struct sockaddr*)&server_addr, sizeof(server_addr)) < 0) {
        perror("Connection failed");
        close(client_socket);
        exit(EXIT_FAILURE);
    }

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

    while (1) {
        // Send message to server
```

Output

```
Connection failed: Connection refused


=== Code Exited With Errors ===
```

main.c

```c

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

    while (1) {
        // Send message to server
        printf("You: ");
        fgets(buffer, BUFFER_SIZE, stdin);
        send(client_socket, buffer, strlen(buffer), 0);

        // Receive response from server
        memset(buffer, 0, BUFFER_SIZE);
        recv(client_socket, buffer, BUFFER_SIZE, 0);
        printf("Server: %s\n", buffer);
    }

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

Output

```
Connection failed: Connect


=== Code Exited With Error
```