

1 .Lab Manual: Developing a TCP Client-Server Application using Linux Socket Programming

Objective:

The objective of this lab is to gain hands-on experience in developing a basic client-server application using TCP sockets in a Linux environment. The lab focuses on establishing a TCP connection, sending a message from the client to the server, and displaying the received message on the server side.

Lab Requirements:

- Linux environment (either a physical machine or a virtual machine).
- Basic knowledge of C programming.

Lab Setup:

1. Setting up the Environment:

- Ensure that you have a Linux environment with a C compiler installed (e.g., gcc).
- Open a terminal.

2. Creating the Project Directory:

- Create a new directory for your project.

```
bash
```

```
2. mkdir tcp_message_lab  
cd tcp_message_lab
```

Part 1: Server Side

Step 1: Writing the Server Code

1. Create a file named server.c in the project directory.

```
c
```

```
1. // server.c  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <arpa/inet.h>  
  
#define PORT 8080  
#define MAX_BUFFER_SIZE 1024  
  
int main() {  
    int server_fd, new_socket, valread;  
    struct sockaddr_in address;  
    int addrlen = sizeof(address);  
    char buffer[MAX_BUFFER_SIZE] = {0};  
  
    // Create a socket  
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {  
        perror("Socket creation failed");  
        exit(EXIT_FAILURE);
```

```

}

// Set up server address struct
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
address.sin_port = htons(PORT);

// Bind the socket to the address
if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
    perror("Bind failed");
    exit(EXIT_FAILURE);
}

// Listen for incoming connections
if (listen(server_fd, 3) < 0) {
    perror("Listen failed");
    exit(EXIT_FAILURE);
}

// Accept incoming connection
if ((new_socket = accept(server_fd, (struct sockaddr *)&address, (socklen_t*)&addrlen)) < 0) {
    perror("Accept failed");
    exit(EXIT_FAILURE);
}

// Read data from the client using TCP
valread = read(new_socket, buffer, MAX_BUFFER_SIZE);
printf("Received message from client: %s\n", buffer);

// Close the connection
close(new_socket);
close(server_fd);

return 0;
}

```

Step 2: Compiling and Running the Server Code

1. Compile the server code.

bash

- gcc server.c -o server
- Run the server.

bash

2. ./server

Part 2: Client Side

Step 1: Writing the Client Code

1. Create a file named client.c in the project directory.

c

1. // client.c

```

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

#define PORT 8080
#define MAX_BUFFER_SIZE 1024

int main() {
    int client_fd;
    struct sockaddr_in server_address;
    char message[MAX_BUFFER_SIZE];

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

    // Configure server address
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(PORT);
    if (inet_pton(AF_INET, "127.0.0.1", &server_address.sin_addr) <= 0) {
        perror("Invalid address/ Address not supported");
        exit(EXIT_FAILURE);
    }

    // Connect to the server using TCP
    if (connect(client_fd, (struct sockaddr *)&server_address, sizeof(server_address)) < 0) {
        perror("Connection Failed");
        exit(EXIT_FAILURE);
    }

    // Get user input for the message
    printf("Enter a message to send to the server: ");
    fgets(message, MAX_BUFFER_SIZE, stdin);

    // Send the message to the server using TCP
    send(client_fd, message, strlen(message), 0);

    // Close the connection
    close(client_fd);

    return 0;
}

```

Step 2: Compiling and Running the Client Code

1. Compile the client code.

bash

- gcc client.c -o client
- Run the client.

bash

2. ./client

In both the server and client code, the `SOCK_STREAM` parameter in the `socket` function indicates the use of TCP. This sets up a reliable, connection-oriented communication channel between the client and the server. The subsequent `read` and `send` functions are used for reading from and writing to the TCP socket, respectively.

OUTPUT:

```
jejo@thinkpad:~/lab$ ./server
Received message from client: hi
```

```
jejo@thinkpad:~/lab$ █
```

```
jejo@thinkpad:~/lab$ ./client cl
Enter a message to send to the server: hi
jejo@thinkpad:~/lab$ █
```

\\

2.Lab Manual: Developing a UDP Client-Server Application using UNIX Socket Programming

Objective:

The objective of this lab is to gain hands-on experience in developing a basic client-server application using UDP sockets in a UNIX environment. The lab focuses on creating a connection between a client and a server, where the client sends a message to the server, and the server displays the received message.

Lab Requirements:

- UNIX-like operating system (Linux or macOS).
- Basic knowledge of C programming.

Lab Setup:

1. Setting up the Environment:

- Ensure that you have a UNIX-like operating system.
- Open a terminal.

2. Creating the Project Directory:

```
bash
```

```
mkdir udp_message_lab  
cd udp_message_lab
```

Part 1: Server Side

Step 1: Writing the Server Code

Create a file named `udp_server.c` in the project directory.

```
c  
  
// udp_server.c  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <string.h>  
#include <arpa/inet.h>  
#define PORT 8080  
#define MAX_BUFFER_SIZE 1024  
  
int main() {  
    // Server code...  
}
```

Step 2: Compiling and Running the Server Code

Compile the server code.

```
bash
```

```
gcc udp_server.c -o udp_server
```

Run the server.

```
bash
```

```
./udp_server
```

Part 2: Client Side

Step 1: Writing the Client Code

Create a file named `udp_client.c` in the project directory.

```
c  
  
// udp_client.c
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <arpa/inet.h>
#define PORT 8080
#define MAX_BUFFER_SIZE 1024

int main() {
    // Client code...
}
```

Step 2: Compiling and Running the Client Code

Compile the client code.

```
bash
```

```
gcc udp_client.c -o udp_client
```

Run the client.

```
bash
```

```
./udp_client
```

Lab Tasks:

1. Server-Client Communication:

- Modify the server and client code to establish a basic communication channel. For example, the client sends a message, and the server receives and prints it.

2. Enhancements:

- Enhance the server-client communication by exchanging more complex data structures or messages.
- Implement error handling to manage unexpected scenarios.

3. Security Considerations:

- Discuss and implement basic security measures (e.g., input validation) in the server and client code.

Explanation:

- **Server Code Explanation:**

- The server code creates a UDP socket using `socket()` with `SOCK_DGRAM`.
- The server binds the socket to a specific address using `bind()`.
- It then waits to receive data from the client using `recvfrom()`.

- **Client Code Explanation:**

- The client code creates a UDP socket using `socket()` with `SOCK_DGRAM`.
- It sends data to the server using `sendto()`.

Conclusion:

This lab provided hands-on experience in creating a simple client-server application using UDP sockets in a UNIX environment. Students are encouraged to explore and enhance the code further, implementing additional features and addressing potential security concerns.

Sample Output:

Run the server and client in separate terminals to observe the communication between them.

Sample output might look like this:

```
bash
```

```
Server: Waiting for client messages...
```

```
Client: Enter a message to send to the server: Hello, server!
```

```
Server: Received message from client: Hello, server!
```

Understanding UDP in the Code:

Server Side:

```
c

// Create a UDP socket
if ((server_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}
```

In the server code, a UDP socket is created using the `socket()` function. The second argument, `SOCK_DGRAM`, specifies that this is a UDP socket.

```
c

// Receive data from the client using UDP
if (recvfrom(server_fd, buffer, MAX_BUFFER_SIZE, 0, (struct sockaddr*)&client_address, &client_address_len) == -1) {
    perror("Receive failed");
    exit(EXIT_FAILURE);
}
```

The `recvfrom()` function is used to receive data from the client. This function is specific to UDP and is used to receive a message from a specific address (in this case, the client's address).

Client Side:

```
c

// Create a UDP socket
if ((client_fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
    perror("Socket creation failed");
    exit(EXIT_FAILURE);
}
```

Similar to the server, the client code creates a UDP socket using the `socket()` function with `SOCK_DGRAM` as the second argument.

c

```
// Send the message to the server using UDP
if (sendto(client_fd, message, strlen(message), 0, (const struct sockaddr*)&server_address, sizeof(server_address)) ==
-1) {
    perror("Send failed");
    exit(EXIT_FAILURE);
}
```

The sendto() function is used to send the message to the server

4.Lab Manual: Analyzing TCP/UDP Performance with NetSim

Objective:

The objective of this lab is to use NetSim for simulating a network and analyzing the performance of TCP and UDP protocols. Participants will gain hands-on experience in creating network scenarios, observing traffic, and analyzing protocol behavior.

Lab Requirements:

- NetSim installed on a Linux system.
- Basic knowledge of networking concepts.

Lab Setup:

Task 0: Installation of NetSim

1. Download NetSim:

- Visit the NetSim download page and download the NetSim installer for Linux.

2. Install NetSim:

- Open a terminal in the directory where the installer is located.
- Run the following commands:

```
bash
```

- # Give execute permissions to the installer

```
chmod +x netsim_linux_installer.sh
```

```
# Run the installer
```

```
sudo ./netsim_linux_installer.sh
```

- Follow the on-screen instructions to complete the installation.

2. Activate License:

- Activate your NetSim license using the provided activation key.

Lab Tasks:

Task 1: Create a Simple Network Topology

1. Open NetSim:

- Launch NetSim on your Linux system.

2. Create a New Project:

- Start a new project in NetSim.

3. Add Devices:

- Use NetSim's graphical interface to add devices such as routers, switches, and end devices to create a simple network topology.

4. Connect Devices:

- Connect the devices using appropriate links and configure basic IP addressing.

bash

```
4. # Example command for device configuration
configure node <Node_ID> interface=<Interface_ID> ip=<IP_Address>
```

Task 2: Configure TCP/UDP Applications

1. Add Applications:

- Integrate TCP and UDP applications into your network devices using NetSim.

bash

```
# Example command for adding TCP application
add application <Node_ID> app=TCP/Telnet
```

bash

```
• # Example command for adding UDP application
add application <Node_ID> app=UDP/FTP
```

• Configure Parameters:

- Set parameters such as packet size, bandwidth, and delay for the TCP and UDP applications.

bash

```
2. # Example command for configuring bandwidth
configure link <Link_ID> bandwidth=<Bandwidth_Value>
```

Task 3: Run Simulation

1. Start Simulation:

- Run the simulation in NetSim.

bash

- # Example command for starting the simulation

run

- **Generate Traffic:**

- Generate traffic by initiating communication between devices.

bash

2. # Example command for generating traffic

start simulation traffic

3. **Observe Results:**

- Use NetSim's simulation results and logs to observe the performance of TCP and UDP applications.

Task 4: Analyze Protocol Behavior

1. **Packet Analysis:**

- Analyze packet traces and logs generated by NetSim to understand how TCP and UDP protocols behave in the simulated environment.

bash

- # Example command for analyzing packets

show packet trace

- **Performance Metrics:**

- Examine key performance metrics such as throughput, latency, and packet loss for both TCP and UDP.

bash

2. # Example command for displaying performance metrics

show simulation metrics

Task 5: Experiment with Network Conditions

1. **Modify Parameters:**

- Experiment with different network conditions by modifying parameters such as bandwidth, delay, and congestion.

bash

1. # Example command for modifying bandwidth

configure link <Link_ID> bandwidth=<New_Bandwidth_Value>

2. **Observe Impact:**

- Observe how changes in network conditions affect the performance of TCP and UDP applications.

Conclusion:

This lab provided hands-on experience in using NetSim to simulate a network, analyze the performance of TCP and UDP protocols, and experiment with different network conditions. Participants gained insights into how these protocols behave under various scenarios.

Notes:

- NetSim offers a wide range of features for detailed network simulations; explore the documentation for additional functionalities.
- Experiment with more complex network topologies and scenarios to deepen your understanding of network simulation.

Sample Output:

- Sample outputs might include simulation logs, graphs, and performance metrics provided by NetSim.

6.Lab Manual: Introduction to Network Tools - ping, traceroute, and netcat

Objective:

The objective of this lab is to introduce and demonstrate the usage of common network tools: ping, traceroute, and netcat. Participants will gain hands-on experience in testing network connectivity, tracing the path of packets, and establishing simple network connections.

Lab Requirements:

- A Linux-based system (Ubuntu, CentOS, etc.) for both client and server.

Lab Setup:

1. Ensure Required Tools:

- Make sure that the required network tools (ping, traceroute, and netcat) are installed on your Linux system. If not, install them using your system's package manager (apt, yum, etc.).

bash

1. # For Ubuntu/Debian

sudo apt-get install iputils-ping traceroute netcat

For CentOS

sudo yum install iputils traceroute nmap-ncat

2. Prepare Server and Client:

- Choose two Linux systems, one as a server and the other as a client. Ensure they are connected on the same network.

Lab Tasks:

Task 1: Ping (PCP)

1. On the Client:

- Open a terminal on the client.

2. Ping the Server:

- Use the ping command to test connectivity to the server.

bash

2. ping <server_ip>

- Observe the round-trip time and packet loss.

Task 2: Traceroute

1. On the Client:

- Open a terminal on the client.

2. Traceroute to the Server:

- Use the traceroute command to trace the path of packets to the server.

bash

2. traceroute <server_ip>

- Analyze the output to understand the route taken by packets to reach the server.

EXECUTION:

```
jejo@thinkpad:~$ traceroute google.com
traceroute to google.com (142.250.193.110), 30 hops max, 60 byte packets
 1  reliance.reliance (192.168.29.1)  6.683 ms  6.652 ms  6.642 ms
 2  10.41.56.1 (10.41.56.1)  14.787 ms  19.020 ms  19.009 ms
 3  172.31.0.148 (172.31.0.148)  18.998 ms  19.381 ms  19.387 ms
 4  192.168.92.70 (192.168.92.70)  22.994 ms  192.168.92.72 (192.168.92.72)  22.985 ms  192.168.92.74 (192.168.92.74)  27.512 ms
 5  172.26.103.69 (172.26.103.69)  22.969 ms  19.345 ms  27.486 ms
 6  172.26.103.131 (172.26.103.131)  22.944 ms  11.704 ms  16.132 ms
 7  192.168.83.28 (192.168.83.28)  11.922 ms  192.168.83.24 (192.168.83.24)  146.908 ms  192.168.83.26 (192.168.83.26)  146.849 ms
 8  * * *
 9  * * *
10  72.14.217.254 (72.14.217.254)  180.985 ms  72.14.196.126 (72.14.196.126)  180.961 ms  72.14.217.254 (72.14.217.254)  180.937 ms
11  * * *
12  74.125.252.214 (74.125.252.214)  180.866 ms  142.251.55.68 (142.251.55.68)  180.849 ms  142.251.55.238 (142.251.55.238)  204.570 ms
13  142.251.55.221 (142.251.55.221)  204.488 ms  108.170.253.122 (108.170.253.122)  204.399 ms  142.251.55.223 (142.251.55.223)  204.377 ms
14  74.125.242.129 (74.125.242.129)  204.362 ms  204.345 ms  maa05s24-in-f14.1e100.net (142.250.193.110)  204.327 ms
```

Task 3: Netcat (NC)

1. On the Server:

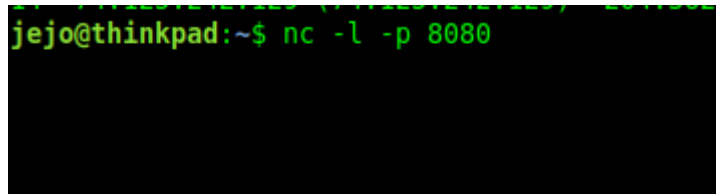
- Open a terminal on the server.

2. Start a Netcat Server:

- Use netcat to listen on a specific port (e.g., 8080).

bash

- `nc -l -p 8080`
- **EXECUTION:**

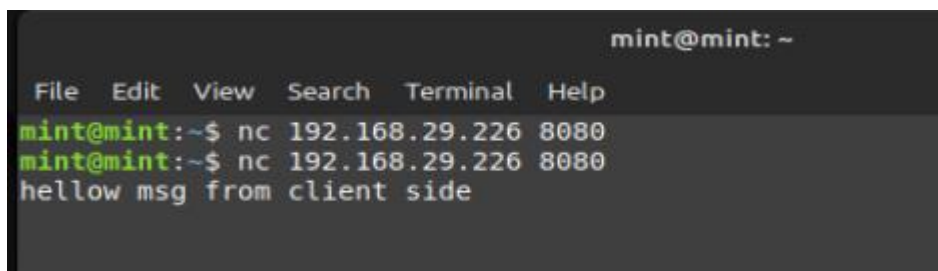
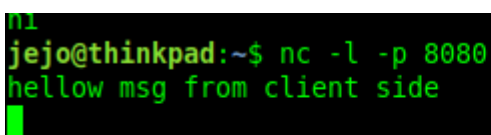
A terminal window on a server with the prompt 'jejo@thinkpad:~\$'. The command 'nc -l -p 8080' has been entered and is shown in green text. The terminal background is black.

- **On the Client:**
- Open a terminal on the client.
- **Establish a Connection:**
- Use netcat to connect to the server's IP and port.

bash

4. `nc <server_ip> 8080`

- Type a message on the client. The message should appear on the server terminal.

A terminal window on a client machine with the prompt 'mint@mint: ~'. It shows a menu with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. Below the menu, the command 'nc 192.168.29.226 8080' is entered twice. The second time, the message 'hellow msg from client side' is received from the server. The terminal background is dark grey.A terminal window on a server with the prompt 'jejo@thinkpad:~\$'. The command 'nc -l -p 8080' has been entered. The message 'hellow msg from client side' is received and shown in green text. The terminal background is black.

Server side:

Conclusion:

This lab introduced basic network tools (ping, traceroute, and netcat) and demonstrated their usage. Participants learned how to test network connectivity, trace the path of packets, and establish simple network connections using these tools.

Notes:

- ping is used for testing basic network connectivity.
- traceroute helps trace the route taken by packets to reach a destination.
- netcat is a versatile tool for network connections, here used to demonstrate a simple client-server connection.

Sample Output:

- Sample outputs might include ping statistics, traceroute path information, and successful netcat connection messages.

7.Lab Manual: Introduction to Packet Analysis with Wireshark

Objective:

The objective of this lab is to introduce participants to the basics of packet analysis using Wireshark. Participants will learn how to capture packets, inspect various protocols, and analyze network traffic.

Lab Requirements:

- A Linux system with Wireshark installed.
- Basic knowledge of networking concepts.

Lab Setup:

1. Install Wireshark:

- Install Wireshark on your Linux system using the package manager.

bash

```
1. # For Ubuntu/Debian
sudo apt-get install wireshark
```

```
# For CentOS
sudo yum install wireshark
```

2. Prepare Network Environment:

- Ensure that your Linux system is connected to a network.

Lab Tasks:

Task 1: Capture Packets

1. Open Wireshark:

- Open Wireshark from the application menu or using the terminal.

bash

1. wireshark

2. Select Network Interface:

- In Wireshark, select the network interface to capture packets.

3. Start Packet Capture:

- Click on the "Start" button to begin capturing packets.

4. Generate Network Traffic:

- Generate some network traffic by browsing a website, sending ping requests, or accessing other network resources.

5. Stop Capture:

- Click on the "Stop" button to end the packet capture.

Task 2: Inspect Captured Packets

1. Analyze Packets:

- In the Wireshark interface, inspect the captured packets.

2. Filtering Packets:

- Use display filters to focus on specific types of packets (e.g., HTTP, TCP, ICMP).

bash

2. # Example: Display only HTTP traffic

http

Task 3: Protocol Analysis

1. Explore Protocols:

- Select a packet in the Wireshark interface and explore the details of various protocols such as Ethernet, IP, TCP, UDP, ICMP, etc.

2. Follow TCP Stream:

- Right-click on a TCP packet and select "Follow" > "TCP Stream" to view the entire TCP stream.

Task 4: Export and Save

1. Export Packets:

- Save specific packets or the entire capture to a file for further analysis.

2. Save as a PCAP File:

- Save the entire capture as a PCAP file.

Task 5: Additional Analysis

1. Statistics:

- Explore the "Statistics" menu to analyze packet statistics, conversations, and more.

2. IO Graphs:

- Use the "Statistics" > "IO Graphs" feature to visualize network activity.

Conclusion:

This lab provided an introduction to packet analysis using Wireshark. Participants learned how to capture packets, inspect protocols, and analyze network traffic. Wireshark is a powerful tool for troubleshooting and understanding network behavior.

Notes:

- Wireshark is a versatile tool; explore various features to deepen your understanding.
- The lab focused on basic packet analysis; Wireshark can be used for more advanced tasks, including security analysis.

Sample Output:

- Sample outputs might include packet details, protocol analysis screenshots, and exported PCAP files.

Additional Resources:

- Wireshark User Guide
- Wireshark Tutorial for Beginners

5,3,-pending sums...