MIT World Peace University

Computer Networks
Second Year B. Tech, Semester 3

UDP Socket Programming

Practical Report
Assignment 9

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A1, PA 20

November 29, 2022

Contents

1 Aim and Objectives

To understand Concept of UDP Socket programming.

2 Problem Statement

Write a C Program for wired network using udp socket to perform Reversing of a String.

3 Platform

Operating System: Arch Linux x86-64
IDEs or Text Editors Used: Visual Studio Code
Programs Used: Cisco Packet Tracer v6.0.1

4 Code

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

void reverse_string(char *str)
{
    int i, j;
    char temp;
    for (i = 0, j = strlen(str) - 1; i < j; i++, j--)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
    }
}

int main(void)
{
    int socket_desc;
    struct sockaddr_in server_addr, client_addr;
    char server_message[2000], client_message[2000];
    int client_struct_length = sizeof(client_addr);

    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));

    // Create UDP socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if (socket_desc < 0)
    {
        printf("Error while creating socket\n");
        return -1;
    }
    printf("Socket created successfully\n");

```

```
    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Bind to the set port and IP:
    if (bind(socket_desc, (struct sockaddr *)&server_addr, sizeof(server_addr)) <
    0)
    {
        printf("Couldn't bind to the port\n");
        return -1;
    }
    printf("Done with binding\n");

    printf("Listening for incoming messages...\n\n");

    // Receive client's message:
    if (recvfrom(socket_desc, client_message, sizeof(client_message), 0,
                 (struct sockaddr *)&client_addr, &client_struct_length) < 0)
    {
        printf("Couldn't receive\n");
        return -1;
    }
    printf("Received message from IP: %s and port: %i\n",
           inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));

    reverse_string(client_message);
    printf("Msg from client: %s\n", client_message);

    // Respond to client:
    strcpy(server_message, client_message);

    if (sendto(socket_desc, server_message, strlen(server_message), 0,
               (struct sockaddr *)&client_addr, client_struct_length) < 0)
    {
        printf("Can't send\n");
        return -1;
    }

    // Close the socket:
    close(socket_desc);

    return 0;
}
```

Listing 1: Server

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

int main(void)
{
    int socket_desc;
    struct sockaddr_in server_addr;
    char server_message[2000], client_message[2000];
    int server_struct_length = sizeof(server_addr);

```

```c
    // Clean buffers:
    memset(server_message, '\0', sizeof(server_message));
    memset(client_message, '\0', sizeof(client_message));

    // Create socket:
    socket_desc = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);

    if (socket_desc < 0)
    {
        printf("Error while creating socket\n");
        return -1;
    }
    printf("Socket created successfully\n");

    // Set port and IP:
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(2000);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Get input from the user:
    printf("Enter message: ");
    gets(client_message);

    // Send the message to server:
    if (sendto(socket_desc, client_message, strlen(client_message), 0,
               (struct sockaddr *)&server_addr, server_struct_length) < 0)
    {
        printf("Unable to send message\n");
        return -1;
    }

    // Receive the server's response:
    if (recvfrom(socket_desc, server_message, sizeof(server_message), 0,
                 (struct sockaddr *)&server_addr, &server_struct_length) < 0)
    {
        printf("Error while receiving server's msg\n");
        return -1;
    }

    printf("Server's response: %s\n", server_message);

    // Close the socket:
    close(socket_desc);

    return 0;
}
```

Listing 2: Client

5 Output

```
SERVER

Socket created successfully
Done with binding
Listening for incoming messages...
```

```
Received message from IP: 127.0.0.1 and port: 34067
Msg from client: olleh

CLIENT

Socket created successfully
Enter message: hello
Server's response: olleh
```

Assignment - 9

UDP Socket Programming

(*) Theory

→ Client / server Communication :

In UDP, the client does not form a connection with the server like in TCP. & instead just sends a datagram. Similarly, the server does not wait for ~~a~~ ~~data~~ accepting a connection and just waits for datagrams to arrive.

Datagrams upon arrival contain the address of the sender which the server uses to send data to the correct client.

→ Introduction to UDP.

UDP is a communication protocol that is primarily used to establish low latency and loss tolerating connections between applications on the Internet.

UDP speeds up transmission by enabling the transfer of data before an agreement is provided by a receiving party.

The UDP Segment Header:

→ UDP wraps datagrams with a segment header which consists of fields totalling 8 bytes. The fields in a UDP header are: Source Port, the port of the device sending the data. This field can be set to zero if the destination computer doesn't need to reply to the sender.

Introduction to Sockets

Sockets are commonly used for client & server interaction. Typical system configuration, places request to server, exchanges information, and then disconnects. A socket has a typical flow of events.

UDP socket functions

The server and client both uses a bind call to associate a local address to the socket. The client can issue an optional bind call to a local address.

The send to and recvfrom calls between client and server are performed until all the data has been transferred. Both the server and client end the session using close() call.

(*) UDP socket flow description on server.

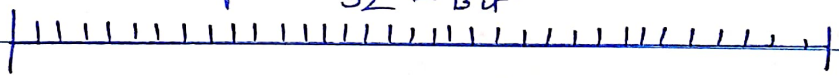
1. Socket()
2. Bind()
3. Receive from()
4. Exit()

(*) UDP socket flow description on client.

1. Socket()
2. Send_to()
3. Recv_from()
4. close()

FAQs

Q.1. Draw and explain UDP header.



Source Port	Destination Port
UDP length	UDP checksum

(*) Difference Between TCP & UDP

	TCP	UDP
1.	Keeps track of lost packets, makes sure lost packets are resent.	Doesnt keep track of lost packets.
2.	Adds sequence number to packets and resends any that arrive in the wrong order.	Doesnt care about packet arrival order.
3.	Slower, because of all the added functionality	Faster, as it lacks any extra features.
4.	Requires more computer resources as OS needs to track ongoing communications and manage them on a much deeper level.	Requires less computer resources
5.	Examples: HTTP, HTTPS, FTP, Computer games.	Eg: DNS IP Telephony DHCP Computer games.