Programming UDP sockets in C

In this lab you are going to learn UDP socket programming. UDP sockets or Datagram sockets are different from the TCP sockets. Unlike TCP, UDP is connection less protocol where we do not need to establish a connection between Client and Server before initiating communication between them. It has low overhead and mostly used in the applications where we are more concerned about delay performance rather than reliability. One obvious example is multimedia communication in which video streaming applications get transport layer services from UDP protocol. In this Lab, you will run client side and server-side code with UDP protocol and understand how to exchange different messages between a client and a server.

The following has been taken from Silver Moon's article published on August 27 2012, on web page with URL: https://www.binarytides.com/programming-udp-sockets-c-linux/. This article describes how to write a simple echo server and client using UDP sockets in C on Linux/Unix platform.

ECHO Server

Let's first make a very simple ECHO server with UDP socket. The flow of the code would be

socket() -> bind() -> recvfrom() -> sendto()

C code

```
/*
    Simple udp server
*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<sys/socket.h>

#define BUFLEN 512  //Max length of buffer
#define PORT 8888   //The port on which to listen for incoming data

void die(char *s)
{
    perror(s);
    exit(1);
}

int main(void)
{
```

```
    struct sockaddr_in si_me, si_other;

    int s, i, slen = sizeof(si_other) , recv_len;
    char buf[BUFLEN];

    //create a UDP socket
    if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
        die("socket");
    }

    // zero out the structure
    memset((char *) &si_me, 0, sizeof(si_me));

    si_me.sin_family = AF_INET;
    si_me.sin_port = htons(PORT);
    si_me.sin_addr.s_addr = htonl(INADDR_ANY);

    //bind socket to port
    if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1)
    {
        die("bind");
    }

    //keep listening for data
    while(1)
    {
        printf("Waiting for data...");
        fflush(stdout);

        //try to receive some data, this is a blocking call
        if ((recv_len = recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other,
&slen)) == -1)
        {
            die("recvfrom()");
        }

        //print details of the client/peer and the data received
        printf("Received packet from %s:%d\n", inet_ntoa(si_other.sin_addr),
ntohs(si_other.sin_port));
        printf("Data: %s\n" , buf);

        //now reply to client with the same data
        if (sendto(s, buf, recv_len, 0, (struct sockaddr*) &si_other, slen) == -1)
        {
            die("sendto()");
        }
    }
```

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

Run the above code by doing a **`gcc server.c && ./a.out`** at the terminal. Then it will show waiting for data like this

```
$ gcc server.c && ./a.out
Waiting for data...
```

Client

The program flow is like

socket() -> sendto()/recvfrom()

Here is a quick example

```
/*
    Simple udp client
*/
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<arpa/inet.h>
#include<sys/socket.h>

#define SERVER "127.0.0.1"
#define BUFLEN 512  //Max length of buffer
#define PORT 8888   //The port on which to send data

void die(char *s)
{
    perror(s);
    exit(1);
}

int main(void)
{
    struct sockaddr_in si_other;
    int s, i, slen=sizeof(si_other);
    char buf[BUFLEN];
    char message[BUFLEN];

    if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1)
    {
        die("socket");
    }
```

```
    memset((char *) &si_other, 0, sizeof(si_other));
    si_other.sin_family = AF_INET;
    si_other.sin_port = htons(PORT);

    if (inet_aton(SERVER , &si_other.sin_addr) == 0)
    {
        fprintf(stderr, "inet_aton() failed\n");
        exit(1);
    }

    while(1)
    {
        printf("Enter message : ");
        gets(message);

        //send the message
        if (sendto(s, message, strlen(message) , 0 , (struct sockaddr *) &si_other,
slen)==-1)
        {
            die("sendto()");
        }

        //receive a reply and print it
        //clear the buffer by filling null, it might have previously received data
        memset(buf,'\0', BUFLEN);
        //try to receive some data, this is a blocking call
        if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen) == -1)
        {
            die("recvfrom()");
        }

        puts(buf);
    }

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

Run the above program and it will ask for some message.

```
$ gcc client.c -o client && ./client
Enter message:
```

Whatever message the client send to server, the same will be echoed.

Task to do:

1. Client reads a line of characters (data) from user and send the data to the server.
2. The server receives the data and converts all the characters to uppercase.
3. Paste your code that convert characters to uppercase.

4. The server send the modified data to the client.
5. The client receives the modified data and displays the line on its screen.

Conclusion: