Programming udp sockets in C on Linux

C By Silver Moon On Aug 27, 2012 2 Comments

UDP sockets

This article describes how to write a simple echo server and client using udp sockets in C on Linux/Unix platform. UDP sockets or Datagram sockets are different from the TCP sockets in a number of ways. The most important difference is that UDP sockets are not connection oriented. More technically speaking, a UDP server does not **accept** connections and a udp client does not **connect** to server.

> The server will bind and then directly receive data and the client shall directly send the data.

ECHO Server

So lets first make a very simple ECHO server with UDP socket. The flow of the code would be

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

C code

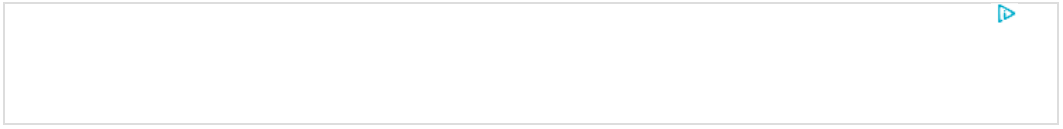
```c
/*
    Simple udp server
    Silver Moon (m00n.silv3r@gmail.com)
*/
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#include<arpa/inet.h>
#include<sys/socket.h>

#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");
    }

```


Related Posts

- ICMP ping flood code using sockets in C –
- Programming raw udp sockets in C on Linu
- Code a simple socket client class in c++
- Receive full data with recv socket function
- Server and client example with C sockets c
- C program to get mac address from interfac on Linux
- C code to perform IP whois
- Handle multiple socket connections with fd_ select on Linux
- Socket programming in C on Linux – tutoria
- C program to get a domain's whois informat using sockets on Linux

```
    //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, &sl
        {
            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_
        printf("Data: %s\n" , buf);

        //now reply the 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...
```

Next step would be to connect to this server using a client. We shall be making a client program a little later but first for testing this code we can use netcat.

Open another terminal and connect to this udp server using netcat and then send some data. The same data will be send back by the server. Over here we are using the ncat command from the nmap package.

```
$ ncat -vv localhost 8888 -u
Ncat: Version 5.21 ( http://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:8888.
hello
hello
world
world
```

Note : We had to use netcat because the ordinary telnet command does not support udp protocol. The -u option of netcat specifies udp protocol.

The **netstat** command can be used to check if the udp port is open or not.

```
$ netstat -u -a
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
udp        0      0 localhost:11211         *:*
udp        0      0 localhost:domain        *:*
udp        0      0 localhost:45286         localhost:8888          ESTABLISHED
udp        0      0 *:33320                 *:*
udp        0      0 *:ipp                   *:*
udp        0      0 *:8888                  *:*
udp        0      0 *:17500                 *:*
udp        0      0 *:mdns                  *:*
udp        0      0 localhost:54747         localhost:54747         ESTABLISHED
udp6       0      0 [::]:60439              [::]:*
udp6       0      0 [::]:mdns               [::]:*
```

Note the *:8888 entry of output. Thats our server program.
The entry that has localhost:8888 in "Foreign Address" column, indicates some client connected to it, which is netcat over here.

Client

Now that we have tested our server with netcat, its time to make a client and use it instead of netcat.
The program flow is like

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

Here is a quick example

```c
/*
    Simple udp client
    Silver Moon (m0On.silv3r@gmail.com)
*/
#include<stdio.h> //printf
#include<string.h> //memset
#include<stdlib.h> //exit(0);
#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, sl
        {
            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 : happy
happy
```

Whatever message the client sends to server, the same comes back as it is and is echoed.

Conclusion

UDP sockets are used by protocols like DNS etc. The main idea behind using UDP is to transfer small amounts of data and where reliability is not a very important issue. UDP is also used in broadcasting/multicasting.

When a file transfer is being done or large amount of data is being transferred in parts the transfer has to be much more reliable for the task to complete. Then the TCP sockets are used.

Last Updated On : 4th January 2013

c sockets socket programming udp sockets

Related Posts

UDP socket programming in winsock
Programming udp sockets in python
Programming raw udp sockets in C on Linux
Server and client example with C sockets on Linux
Socket programming in C on Linux – tutorial

About **Silver Moon**

Php developer, blogger and Linux enthusiast. He can be reached at admin@binarytides.com. Or find him on Google+