C Socket Programming for Linux with a Server and Client Example Code

by Himanshu Arora on December 19, 2011

Typically two processes communicate with each other on a single system through one of the following inter process communication techniques.

- Pipes
- Message queues
- Shared memory

There are several other methods. But the above are some of the very classic ways of interprocess communication.

But have you ever given a thought over how two processes communicate across a network?

For example, when you browse a website, on your local system the process running is your web browser, while on the remote system the process running is the web server. So this is also an inter process communication but the technique through which they communicate with each other is SOCKETS, which is the focus of this article.

What is a SOCKET?

In layman's term, a Socket is an end point of communication between two systems on a network. To be a bit precise, a socket is a combination of IP address and port on one system. So on each system a socket exists for a process interacting with the socket on other system over the network. A combination of local socket and the socket at the remote system is also known a 'Four tuple' or '4-tuple'. Each connection between two processes running at different systems can be uniquely identified through their 4-tuple.

There are two types of network communication models:

1. OSI
2. TCP/IP

While OSI is more of a theoretical model, the TCP/IP networking model is the most popular and widely used.

As explained in our TCP/IP Fundamentals article, the communication over the network in TCP/IP model takes place in form of a client server architecture. ie, the client begins the communication and server follows up and a connection is established.

Sockets can be used in many languages like Java, C++ etc but here in this article, we will understand the socket communication in its purest form (i.e in C programming language)

Lets create a server that continuously runs and sends the date and time as soon as a client connects to it.

Socket Server Example

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <time.h>

int main(int argc, char *argv[])
{
    int listenfd = 0, connfd = 0;
    struct sockaddr_in serv_addr;

    char sendBuff[1025];
    time_t ticks;

    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    memset(&serv_addr, '0', sizeof(serv_addr));
    memset(sendBuff, '0', sizeof(sendBuff));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port = htons(5000);

    bind(listenfd, (struct sockaddr*)&serv_addr, sizeof(serv_addr));

    listen(listenfd, 10);

    while(1)
    {
        connfd = accept(listenfd, (struct sockaddr*)NULL, NULL);

        ticks = time(NULL);
        snprintf(sendBuff, sizeof(sendBuff), "%.24s\r\n", ctime(&ticks));
        write(connfd, sendBuff, strlen(sendBuff));

        close(connfd);
        sleep(1);
     }
```

```
}
```

In the above program, we have created a server. In the code :

- The call to the function ‘socket()’ creates an UN-named socket inside the kernel and returns an integer known as socket descriptor.
- This function takes domain/family as its first argument. For Internet family of IPv4 addresses we use AF_INET.
- The second argument ‘SOCK_STREAM’ specifies that the transport layer protocol that we want should be reliable ie it should have acknowledgement techniques. For example : TCP
- The third argument is generally left zero to let the kernel decide the default protocol to use for this connection. For connection oriented reliable connections, the default protocol used is TCP.
- The call to the function ‘bind()’ assigns the details specified in the structure ‘serv_addr’ to the socket created in the step above. The details include, the family/domain, the interface to listen on(in case the system has multiple interfaces to network) and the port on which the server will wait for the client requests to come.
- The call to the function ‘listen()’ with second argument as ’10′ specifies maximum number of client connections that server will queue for this listening socket.
- After the call to listen(), this socket becomes a fully functional listening socket.
- In the call to accept(), the server is put to sleep and when for an incoming client request, the three way TCP handshake* is complete, the function accept () wakes up and returns the socket descriptor representing the client socket.
- The call to accept() is run in an infinite loop so that the server is always running and the delay or sleep of 1 sec ensures that this server does not eat up all of your CPU processing.
- As soon as server gets a request from client, it prepares the date and time and writes on the client socket through the descriptor returned by accept().

Three way handshake is the procedure that is followed to establish a TCP connection between two remote hosts. We might soon be posting an article on the theoretical aspect of the TCP protocol.

Finally, we compile the code and run the server.

Socket Client Example

```
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <arpa/inet.h>

int main(int argc, char *argv[])
```

```
{
    int sockfd = 0, n = 0;
    char recvBuff[1024];
    struct sockaddr_in serv_addr;

    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }

    memset(recvBuff, '0',sizeof(recvBuff));
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Error : Could not create socket \n");
        return 1;
    }

    memset(&serv_addr, '0', sizeof(serv_addr));

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5000);

    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occured\n");
        return 1;
    }

    if( connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))< 0)
    {
       printf("\n Error : Connect Failed \n");
       return 1;
    }

    while ( (n = read(sockfd, recvBuff, sizeof(recvBuff)-1)) > 0)
    {
        recvBuff[n] = 0;
        if(fputs(recvBuff, stdout) == EOF)
        {
            printf("\n Error : Fputs error\n");
        }
    }

    if(n < 0)
    {
        printf("\n Read error \n");
    }

    return 0;
}
```

In the above program, we create a client which will connect to the server and receive date and time from it. In the above piece of code :

- We see that here also, a socket is created through call to socket() function.

- Information like IP address of the remote host and its port is bundled up in a structure and a call to function connect() is made which tries to connect this socket with the socket (IP address and port) of the remote host.
- Note that here we have not bind our client socket on a particular port as client generally use port assigned by kernel as client can have its socket associated with any port but In case of server it has to be a well known socket, so known servers bind to a specific port like HTTP server runs on port 80 etc while there is no such restrictions on clients.
- Once the sockets are connected, the server sends the data (date+time) on clients socket through clients socket descriptor and client can read it through normal read call on the its socket descriptor.

Now execute the client as shown below.

```
$ ./newsc 127.0.0.1
Sun Dec  18 22:22:14 2011
```

We can see that we successfully got the date and time from server. We need to send the IP address of the server as an argument for this example to run. If you are running both server and client example on the same machine for testing purpose, use the loop back ip address as shown above.

To conclude, In this article we studied the basics of socket programming through a live example that demonstrated communication between a client and server processes capable of running on two different machines.