Unix Network Programming Tutorials - Socket Basics

You might have studied various mechanisms through which two processes interact on same host. For example message queues, shared memory etc. But what about interaction between two process running on two different hosts (lets say on a LAN or WAN). Here comes the concept of network programming where in two processes running on different hosts interact through sockets.

Now, what is a socket?
An end point of Inter process communication between two processes running on different hosts is known as a Socket. Basically Socket is a term used for combination of an IP address and a port on which a process communicates with any other process on network.

Example

As always, lets consider an example where two processes communicate with each other. In this case, two processes are running on same host, as I don't have two separate machines :-) . But nevertheless, it will demonstrate the usage of sockets in equally good way.

There is a server which constantly runs and waits for a client to connect to it. As soon as a client connects,the server sends date and time to the client.

Ok, wait is over. Lets look at the code. Here comes the server :

Code:

```
#include <sys/types.h>
#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<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 :

- The socket() function creates an internet stream socket. This function returns a small integer that is used to identify the socket in all future socket function calls. Then we fill the socket address structure with the port that we want our server to run on. In this case we assigned port number 500 for the server. Now, for IP we have assigned INADDR_ANY which allows the server to accept a client connection on any network interface(in case the server host has multiple interfaces).

- Through Bind() the server is bound to this socket, hence on a network the identity of this server is this socket.

- Now, through listen() the socket is converted to listening socket on which the incoming connections will be accepted by the kernel.

- So, the steps socket,bind, listen are used to create a listening socket.

- The constant '10' in the call to listen() specifies maximum number of client connections that server will queue for this listening socket.

- Now, during call to accept the server is put to sleep and when a three way TCP handshake is completed by kernel for an incoming connection, then accept() function returns the connected descriptor. This descriptor can be used to communicate with the client. For each client a new connected descriptor is returned by accept().

- The accept() is run under an infinite while loop as server runs forever while a sleep of 1 sec ensures that this loop doesn't make the system slow.

- Now we start this server.

Here comes a client :

Code:

```
#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;
}
```

Now, in the above client code we see that client also creates a socket, fills in the port of server and IP address of server(passed as argument to this client). The function inet_pton() converts the IP address from the presentation (dotted) form to the form in which it can be transfered on network. Through connect() the client establishes connection with server. Through multiple read() we read the data (in our case the date/time) until all the data is read and push the output on stdout.

Now, while the server is already running, lets run the client.
Here is the output :

~/practice $ ./socket_cli 127.0.0.1
Sun Sep 25 12:05:40 2011

Hence we see that the output from server is received and printed by client.

Conclusion

So to conclude, sockets are very popular for Inter process communications mainly for the process running on network. There are some deficiencies in our program like the user must enter the server's IP address as a dotted-decimal number. People work better with names instead of numbers (e.g.,www.google.com). So things like that. But I think an introduction of the level of this simplicity was required. We will discuss more intricacies in the coming part.

Source: http://www.go4expert.com/articles/unix-network-programming-tutorials-t26786/