NETWORK PROGRAMMING LINUX SOCKET PART 11: TCP CLIENT-SERVER CODE SAMPLE

Working program examples if any compiled using gcc, tested using the public IPs, run on **Linux Fedora 3** with several times update, as normal user. The Fedora machine used for the testing having the "No Stack Execute" disabled and the SELinux set to default configuration. All the program example is generic. Beware codes that expand more than one line.

Example: Connecting a TCP client to a server, a client program

- Well, let try the client program that will connect to the previous server program.
- The following example shows how to connect a client socket program to a connection-oriented server.

```
/************tcpclient.c************************/
/* Header files needed to use the sockets API. */
/* File contains Macro, Data Type and */
/* Structure definitions along with Function */
/* prototypes. */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
```

```
#include <errno.h>
/* BufferLength is 100 bytes */
#define BufferLength 100
/* Default host name of server system. Change it
to your default */
/* server hostname or IP.  If the user do not
supply the hostname */
/* as an argument, the_server_name_or_IP will be
used as default*/
#define SERVER "The_server_name_or_IP"
/* Server's port number */
#define SERVPORT 3111

/* Pass in 1 parameter which is either the */
/* address or host name of the server, or */
/* set the server name in the #define SERVER ...
*/
int main(int argc, char *argv[])
{
/* Variable and structure definitions. */
int sd, rc, length = sizeof(int);
struct sockaddr_in serveraddr;
char buffer[BufferLength];
char server[255];
char temp;
int totalcnt = 0;
struct hostent *hostp;
char data[100] = "This is a test string from
client lol!!! ";

/* The socket() function returns a socket */
/* descriptor representing an endpoint. */
/* The statement also identifies that the */


/* INET (Internet Protocol) address family */
/* with the TCP transport (SOCK_STREAM) */
/* will be used for this socket. */
/******************************************/
/* get a socket descriptor */
if((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
{
perror("Client-socket() error");
exit(-1);
}
else
printf("Client-socket() OK\n");
```

```
/*If the server hostname is supplied*/
if(argc > 1)
{
/*Use the supplied argument*/
strcpy(server, argv[1]);
printf("Connecting to the f***ing %s, port %d ...\n",
server, SERVPORT);
}
else
/*Use the default server name or IP*/
strcpy(server, SERVER);

memset(&serveraddr, 0x00, sizeof(struct
sockaddr_in));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(SERVPORT);

if((serveraddr.sin_addr.s_addr = inet_addr(server))
== (unsigned long)INADDR_NONE)
{

/* When passing the host name of the server as a */
/* parameter to this program, use the gethostbyname()
*/
/* function to retrieve the address of the host
server. */
/***************************************************/
/* get host address */
hostp = gethostbyname(server);
if(hostp == (struct hostent *)NULL)
{
printf("HOST NOT FOUND --> ");
/* h_errno is usually defined */
/* in netdb.h */
printf("h_errno = %d\n",h_errno);
printf("---This is a client program---\n");
printf("Command usage: %s <server name or IP>\n",
argv[0]);
close(sd);
exit(-1);
}
memcpy(&serveraddr.sin_addr, hostp->h_addr,
sizeof(serveraddr.sin_addr));
}

/* After the socket descriptor is received, the */
/* connect() function is used to establish a */
/* connection to the server. */
/***********************************************/
/* connect() to server. */
if((rc = connect(sd, (struct sockaddr *)&serveraddr,
sizeof(serveraddr))) < 0)
{
perror("Client-connect() error");
close(sd);
exit(-1);
}
else
printf("Connection established...\n");

/* Send string to the server using */
/* the write() function. */
/*********************************************/
/* Write() some string to the server. */
printf("Sending some string to the f***ing %s...\n",
server);
rc = write(sd, data, sizeof(data));

if(rc < 0)
{
perror("Client-write() error");
rc = getsockopt(sd, SOL_SOCKET, SO_ERROR, &temp,
&length);
if(rc == 0)
{
/* Print out the asynchronously received error. */
errno = temp;
perror("SO_ERROR was");
}
close(sd);
exit(-1);
}
else
{
printf("Client-write() is OK\n");
printf("String successfully sent lol!\n");
printf("Waiting the %s to echo back...\n", server);
}

totalcnt = 0;
while(totalcnt < BufferLength)
{

/* Wait for the server to echo the */
/* string by using the read() function. */
/***************************************/
/* Read data from the server. */
rc = read(sd, &buffer[totalcnt], BufferLength-
totalcnt);
if(rc < 0)
{
perror("Client-read() error");
close(sd);
exit(-1);
}
else if (rc == 0)
{
printf("Server program has issued a close()\n");
close(sd);
exit(-1);
}
else
totalcnt += rc;
}
printf("Client-read() is OK\n");
printf("Echoed data from the f***ing server: %s\n",
buffer);

/* When the data has been read, close() */
/* the socket descriptor. */
/****************************************/
/* Close socket descriptor from client side. */
close(sd);
exit(0);
return 0;
}
```

- Compile and link the client program.

```
[bodo@bakawali testsocket]$ gcc -g tcpclient.c -o
tcpclient
```

- Run the program. Before that don't forget to run the server program first. The first run is without the server hostname/IP.

```
[bodo@bakawali testsocket]$ ./tcpclient
Client-socket() OK
HOST NOT FOUND --> h_errno = 1
---This is a client program---
Command usage: ./tcpclient <server name or IP>
[bodo@bakawali testsocket]$
```

- Then run with the server hostname or IP.

```
[bodo@bakawali testsocket]$ ./tcpclient 203.106.93.94
Client-socket() OK
Connecting to the f***ing 203.106.93.94, port 3111
...
Connection established...
Sending some string to the f***ing 203.106.93.94...
Client-write() is OK
String successfully sent lol!
Waiting the 203.106.93.94 to echo back...
Client-read() is OK
Echoed data from the f***ing server: This is a test
string from client lol!!!
[bodo@bakawali testsocket]$
```

- And at the server console messages.

```
[bodo@bakawali testsocket]$ ./tcpserver
Server-socket() is OK
Server-setsockopt() is OK
Using 0.0.0.0, listening at 3111
Server-bind() is OK
Server-Ready for client connection...
Server-accept() is OK
Server-new socket, sd2 is OK...
Got connection from the f***ing client: 203.106.93.94
Server-read() is OK
Received data from the f***ing client: This is a test
string from client lol!!!
Server-Echoing back to client...
[bodo@bakawali testsocket]$
```

- Well, it works!

UDP connectionless client/server

- The connectionless protocol server and client examples illustrate the socket APIs that are written for User Datagram Protocol (UDP). The server and client examples use the following sequence of function calls:

  1. socket()
  2. bind()

- The following figure illustrates the client/server relationship of the socket APIs for a connectionless protocol.

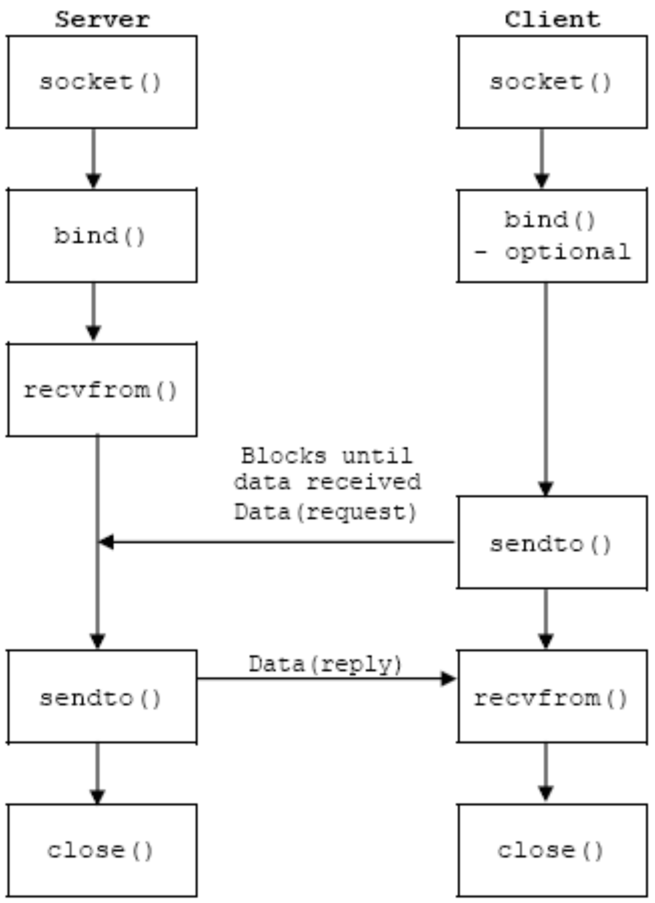

Figure 1: UDP connectionless APIs relationship.

Connecting a UDP server and client

- The following examples show how to use UDP to connect a server to a connectionless client, and a connectionless client to a server.

Example: Connecting a UDP server to a client, a server program

- The first example shows how to use UDP to connect a connectionless server socket program to a client.

```
/*******************udpserver.c*****************/
/* Header files needed to use the sockets API. */
/* File contain Macro, Data Type and Structure */
/* definitions along with Function prototypes. */
/* header files */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>
#include <arpa/inet.h>
/* Server's port number, listen at 3333 */
#define SERVPORT 3333

/* Run the server without argument */
int main(int argc, char *argv[])
{
/* Variable and structure definitions. */
int sd, rc;
struct sockaddr_in serveraddr, clientaddr;
int clientaddrlen = sizeof(clientaddr);
int serveraddrlen = sizeof(serveraddr);
char buffer[100];
char *bufptr = buffer;
int buflen = sizeof(buffer);

/* The socket() function returns a socket */
/* descriptor representing an endpoint. */
/* The statement also identifies that the */
/* INET (Internet Protocol) address family */
/* with the UDP transport (SOCK_DGRAM) will */
/* be used for this socket. */
/******************************************/
/* get a socket descriptor */
if((sd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
perror("UDP server - socket() error");
exit(-1);
}
else
printf("UDP server - socket() is OK\n");

printf("UDP server - try to bind...\n");

/* After the socket descriptor is received, */
/* a bind() is done to assign a unique name */
/* to the socket. In this example, the user */
/* set the s_addr to zero. This allows the */
/* system to connect to any client that uses */
/* port 3333. */
/********************************************/
/* bind to address */
memset(&serveraddr, 0x00, serveraddrlen);
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(SERVPORT);
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
if((rc = bind(sd, (struct sockaddr *)&serveraddr,
serveraddrlen)) < 0)
{
perror("UDP server - bind() error");
close(sd);
/* If something wrong with socket(), just exit lol */
exit(-1);
}
else
printf("UDP server - bind() is OK\n");

printf("Using IP %s and port %d\n",
inet_ntoa(serveraddr.sin_addr), SERVPORT);
printf("UDP server - Listening...\n");

/* Use the recvfrom() function to receive the */
/* data. The recvfrom() function waits */
/* indefinitely for data to arrive. */
/************************************************/
/* This example does not use flags that control */
/* the reception of the data. */
/************************************************/
/* Wait on client requests. */
rc = recvfrom(sd, bufptr, buflen, 0, (struct sockaddr
*)&clientaddr, &clientaddrlen);
if(rc < 0)
{
perror("UDP Server - recvfrom() error");
close(sd);
exit(-1);
}
else
printf("UDP Server - recvfrom() is OK...\n");

printf("UDP Server received the following:\n \"%s\"
message\n", bufptr);
printf("from port %d and address %s.\n",
ntohs(clientaddr.sin_port),
inet_ntoa(clientaddr.sin_addr));

/* Send a reply by using the sendto() function. */
/* In this example, the system echoes the received */
/* data back to the client. */
/************************************************/
/* This example does not use flags that control */
/* the transmission of the data */
/************************************************/
```

```
/* Send a reply, just echo the request */
printf("UDP Server replying to the stupid UDP
client...\n");
rc = sendto(sd, bufptr, buflen, 0, (struct sockaddr
*)&clientaddr, clientaddrlen);
if(rc < 0)
{
perror("UDP server - sendto() error");
close(sd);
exit(-1);
}
else
printf("UDP Server - sendto() is OK...\n");

/* When the data has been sent, close() the */
/* socket descriptor. */
/********************************************/
/* close() the socket descriptor. */
close(sd);
exit(0);
}
```

- Compile and link the udp server program.

```
[bodo@bakawali testsocket]$ gcc -g udpserver.c -o
udpserver
```

- Run the program and let it run in the background.

```
[bodo@bakawali testsocket]$ ./udpserver
UDP server - socket() is OK
UDP server - try to bind...
UDP server - bind() is OK
Using IP 0.0.0.0 and port 3333
UDP server - Listening...

[1]+  Stopped                 ./udpserver
[bodo@bakawali testsocket]$ bg
[1]+ ./udpserver &
[bodo@bakawali testsocket]$
```

- Verify the program running.

```
[bodo@bakawali testsocket]$ ps aux | grep udpserver
bodo      7963  0.0  0.2  2240  324 pts/2    S
12:22   0:00 ./udpserver
bodo      7965  0.0  0.5  4324  648 pts/2    S+
12:24   0:00 grep udpserver
```

- Verify that the udp server is listening at port 3333 waiting for the client connection.

```
[bodo@bakawali testsocket]$ netstat -a | grep 3333
udp        0      0 *:3333                      *:*
[bodo@bakawali testsocket]$
```

- Without the client program (next example) you can try telneting the server using port 3333 for testing. For this program example the following telnet session cannot be established for UDP/connectionless.

```
[bodo@bakawali testsocket]$ telnet 203.106.93.94 3333
Trying 203.106.93.94...
telnet: connect to address 203.106.93.94: Connection
refused
telnet: Unable to connect to remote host: Connection
refused
```

Example: Connecting a UDP client to a server, a client program

- The following example shows how to use UDP to connect a connectionless client socket program to a server. This program will be used to connect to the previous UDP server.

```
/****************udpclient.c********************/
/* Header files needed to use the sockets API. */
/* File contain Macro, Data Type and Structure */
/* definitions along with Function prototypes. */
/***********************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
```

```
#include <arpa/inet.h>
#include <netdb.h>

/* Host name of my system, change accordingly */
/* Put the server hostname that run the UDP server
program */
/* This will be used as default UDP server for client
connection */
#define SERVER "bakawali"
/* Server's port number */
#define SERVPORT 3333
/* Pass in 1 argument (argv[1]) which is either the
*/
/* address or host name of the server, or */
/* set the server name in the #define SERVER above.
*/

int main(int argc, char *argv[])
{
/* Variable and structure definitions. */
int sd, rc;
struct sockaddr_in serveraddr, clientaddr;
int serveraddrlen = sizeof(serveraddr);
char server[255];
char buffer[100];
char *bufptr = buffer;
int buflen = sizeof(buffer);
struct hostent *hostp;
memset(buffer, 0x00, sizeof(buffer));
/* 36 characters + terminating NULL */
memcpy(buffer, "Hello! A client request message
lol!", 37);

/* The socket() function returns a socket */
/* descriptor representing an endpoint. */
/* The statement also identifies that the */
/* INET (Internet Protocol) address family */
/* with the UDP transport (SOCK_DGRAM) will */
/* be used for this socket. */
/******************************************/
/* get a socket descriptor */
if((sd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
{
perror("UDP Client - socket() error");
/* Just exit lol! */
exit(-1);
}
```

```
else
printf("UDP Client - socket() is OK!\n");

/* If the hostname/IP of the server is supplied */
/* Or if(argc = 2) */
if(argc > 1)
strcpy(server, argv[1]);
else
{
/*Use default hostname or IP*/
printf("UDP Client - Usage %s <Server hostname or
IP>\n", argv[0]);
printf("UDP Client - Using default hostname/IP!\n");

strcpy(server, SERVER);
}

memset(&serveraddr, 0x00, sizeof(struct
sockaddr_in));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(SERVPORT);

if((serveraddr.sin_addr.s_addr = inet_addr(server))
== (unsigned long)INADDR_NONE)
{
/* Use the gethostbyname() function to retrieve */
/* the address of the host server if the system */
/* passed the host name of the server as a parameter.
*/
/**********************************************/
/* get server address */
hostp = gethostbyname(server);
if(hostp == (struct hostent *)NULL)
{
printf("HOST NOT FOUND --> ");
/* h_errno is usually defined */
/* in netdb.h */
printf("h_errno = %d\n", h_errno);
exit(-1);
}
else
{
printf("UDP Client - gethostname() of the server is
OK... \n");
printf("Connected to UDP server %s on port %d.\n",
server, SERVPORT);
}
```

```
memcpy(&serveraddr.sin_addr, hostp->h_addr,
sizeof(serveraddr.sin_addr));
}

/* Use the sendto() function to send the data */
/* to the server. */
/************************************************/
/* This example does not use flags that control */
/* the transmission of the data. */
/************************************************/
/* send request to server */
rc = sendto(sd, bufptr, buflen, 0, (struct sockaddr
*)&serveraddr, sizeof(serveraddr));
if(rc < 0)
{
perror("UDP Client - sendto() error");
close(sd);
exit(-1);
}
else
printf("UDP Client - sendto() is OK!\n");

printf("Waiting a reply from UDP server...\n");

/* Use the recvfrom() function to receive the */
/* data back from the server. */
/************************************************/
/* This example does not use flags that control */
/* the reception of the data. */
/************************************************/
/* Read server reply. */
/* Note: serveraddr is reset on the recvfrom()
function. */
rc = recvfrom(sd, bufptr, buflen, 0, (struct sockaddr
*)&serveraddr, &serveraddrlen);

if(rc < 0)
{
perror("UDP Client - recvfrom() error");
close(sd);
exit(-1);
}
else
{
printf("UDP client received the following: \"%s\"
message\n", bufptr);
printf(" from port %d, address %s\n",
ntohs(serveraddr.sin_port),
inet_ntoa(serveraddr.sin_addr));
}

/* When the data has been received, close() */
/* the socket descriptor. */
/********************************************/
/* close() the socket descriptor. */
close(sd);
exit(0);
}
```

- Compile and link the program.

```
[bodo@bakawali testsocket]$ gcc -g udpclient.c -o
udpclient
```

- Run the program. Before that make sure the previous program example (the UDP server) is running.

```
[bodo@bakawali testsocket]$ ./udpclient
UDP Client - socket() is OK!
UDP Client - Usage ./udpclient <Server hostname or
IP>
UDP Client - Using default hostname/IP!
UDP Client - gethostname() of the server is OK...
Connected to UDP server bakawali on port 3333.
UDP Client - sendto() is OK!
Waiting a reply from UDP server...
UDP client received the following: "Hello! A client
request message lol!" message
 from port 3333, address 203.106.93.94
[bodo@bakawali testsocket]$
```

- Well, our udp server and client communicated successfully. The following are the expected messages at server console.

```
[bodo@bakawali testsocket]$ ./udpserver
UDP server - socket() is OK
UDP server - try to bind...
UDP server - bind() is OK
Using IP 0.0.0.0 and port 3333
UDP server - Listening...
UDP Server - recvfrom() is OK...
UDP Server received the following:
 "Hello! A client request message lol!" message
from port 32824 and address 203.106.93.94.
UDP Server replying to the stupid UDP client...
UDP Server - sendto() is OK...
[bodo@bakawali testsocket]$
```

Continue on next Module…TCP/IP and RAW socket, more program examples.

Further reading and digging:

1. Check the best selling C/C++, Networking, Linux and Open Source books at Amazon.com.
2. Broadcasting.
3. Telephony.
4. GCC, GDB and other related tools.

| Winsock & .NET | Winsock | < More TCP, UDP, Client-Server Examples | Linux Socket Index | Iterative Server Program Example > |