NETWORK PROGRAMMING SOCKET PART 10 - MORE TCP & UDP CLIENT & SERVER PROGRAM EXAMPLES

This is a continuation from Part II series, Socket Part 9. 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. Have a nice ride lol!

This Module will cover the following sub-topics:

- Example: select() server
- Connecting a TCP server and client:
  1. Example: Connecting a TCP server to a client, a server program
  2. Example: Connecting a TCP client to a server, a client program
- UDP connectionless client/server
- Connecting a UDP server and client:
  1. Example: Connecting a UDP server to a client, a server program
  2. Example: Connecting a UDP client to a server, a client program
- Connection-oriented server designs:

Example: The select() server

- The following program example acts like a simple multi-user chat server. Start running it in one window, then telnet to it ("`telnet hostname 2020`") from other windows.
- When you type something in one telnet session, it should appear in all the others windows.

```
/ *******select.c*********/
/ *******Using select() for I/O
multiplexing */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
/* port we're listening on */
#define PORT 2020

int main(int argc, char *argv[])
{
/* master file descriptor list */
fd_set master;
/* temp file descriptor list for select()
*/
fd_set read_fds;
/* server address */
struct sockaddr_in serveraddr;


/* client address */
struct sockaddr_in clientaddr;
/* maximum file descriptor number */
int fdmax;
```

```
/* listening socket descriptor */
int listener;
/* newly accept()ed socket descriptor */
int newfd;
/* buffer for client data */
char buf[1024];
int nbytes;
/* for setsockopt() SO_REUSEADDR, below */
int yes = 1;
int addrlen;
int i, j;
/* clear the master and temp sets */
FD_ZERO(&master);
FD_ZERO(&read_fds);

/* get the listener */
if((listener = socket(AF_INET, SOCK_STREAM, 0)) ==
-1)
{
perror("Server-socket() error lol!");
/*just exit lol!*/
exit(1);
}
printf("Server-socket() is OK...\n");
/*"address already in use" error message */
if(setsockopt(listener, SOL_SOCKET, SO_REUSEADDR,
&yes, sizeof(int)) == -1)
{
perror("Server-setsockopt() error lol!");
exit(1);
}
printf("Server-setsockopt() is OK...\n");

/* bind */
serveraddr.sin_family = AF_INET;
serveraddr.sin_addr.s_addr = INADDR_ANY;
serveraddr.sin_port = htons(PORT);
memset(&(serveraddr.sin_zero), '\0', 8);

if(bind(listener, (struct sockaddr *)&serveraddr,
sizeof(serveraddr)) == -1)
{
    perror("Server-bind() error lol!");
    exit(1);
}
printf("Server-bind() is OK...\n");
```

```
/* listen */
if(listen(listener, 10) == -1)
{
     perror("Server-listen() error lol!");
     exit(1);
}
printf("Server-listen() is OK...\n");

/* add the listener to the master set */
FD_SET(listener, &master);
/* keep track of the biggest file descriptor */
fdmax = listener; /* so far, it's this one*/

/* loop */
for(;;)
{
/* copy it */
read_fds = master;

if(select(fdmax+1, &read_fds, NULL, NULL, NULL) ==
-1)
{
    perror("Server-select() error lol!");
    exit(1);
}
printf("Server-select() is OK...\n");

/*run through the existing connections looking for
data to be read*/
for(i = 0; i <= fdmax; i++)
{
    if(FD_ISSET(i, &read_fds))
    { /* we got one... */
    if(i == listener)
     {
         /* handle new connections */
        addrlen = sizeof(clientaddr);
if((newfd = accept(listener, (struct sockaddr
*)&clientaddr, &addrlen)) == -1)
{
    perror("Server-accept() error lol!");
}
else
{
    printf("Server-accept() is OK...\n");

FD_SET(newfd, &master); /* add to master set */
```

```
if(newfd > fdmax)
{ /* keep track of the maximum */
    fdmax = newfd;
}
printf("%s: New connection from %s on socket %d\n",
argv[0], inet_ntoa(clientaddr.sin_addr), newfd);
}
}
else
{
/* handle data from a client */
if((nbytes = recv(i, buf, sizeof(buf), 0)) <= 0)
{
/* got error or connection closed by client */
if(nbytes == 0)
 /* connection closed */
 printf("%s: socket %d hung up\n", argv[0], i);

else
perror("recv() error lol!");

/* close it... */
close(i);
/* remove from master set */
FD_CLR(i, &master);
}
else
{
/* we got some data from a client*/
for(j = 0; j <= fdmax; j++)
{
/* send to everyone! */
if(FD_ISSET(j, &master))
{
       /* except the listener and ourselves */
       if(j != listener && j != i)
       {
              if(send(j, buf, nbytes, 0) == -1)
                     perror("send() error lol!");
       }
}
}
}
}
}
}
}
```

```
return 0;
}
```

- Compile and link the program.

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

- Run the program.

```
[bodo@bakawali testsocket]$ ./select
Server-socket() is OK...
Server-setsockopt() is OK...
Server-bind() is OK...
Server-listen() is OK...
```

- You can leave the program running at the background (Ctrl + z).

```
[bodo@bakawali testsocket]$ ./select
Server-socket() is OK...
Server-setsockopt() is OK...
Server-bind() is OK...
Server-listen() is OK...

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

- Do some verification.

```
[bodo@bakawali testsocket]$ ps aux | grep select
bodo     27474  0.0  0.2  1384  292 pts/2    S+
14:32   0:00 ./select
bodo     27507  0.0  0.5  3724  668 pts/3    S+
14:34   0:00 grep select
[bodo@bakawali testsocket]$ netstat -a |grep 2020
tcp        0      0 *:2020    *:*
LISTEN
[bodo@bakawali testsocket]$
```

- Telnet from other computers or windows using hostname or the IP address. Here we use hostname, bakawali. Use escape character (Ctrl +]) to terminate command. For other telnet command please type help.

```
[bodo@bakawali testsocket]$ telnet bakawali 2020
Trying 203.106.93.94...
Connected to bakawali.jmti.gov.my (203.106.93.94).
```

```
Escape character is '^]'.
^]
telnet> mode line
testing some text
the most visible one
```

- The last two messages were typed at another two machines that connected through socket 5 and 6 (socket 4 is another window of the server) using telnet. Socket 5 and 6 are from Windows 2000 Server machines.

- The following are messages on the server console. There are another two machine connected to the server and the messages at the server console is shown below.

```
[bodo@bakawali testsocket]$ Server-select() is OK...
Server-accept() is OK...
./select: New connection from 203.106.93.94 on socket
4
Server-select() is OK...
...
Server-accept() is OK...
./select: New connection from 203.106.93.91 on socket
5
Server-select() is OK...
Server-select() is OK...
...
Server-select() is OK...
Server-select() is OK...
Server-accept() is OK...
./select: New connection from 203.106.93.82 on socket
6
```

- When the clients disconnected from the server through socket 4, 5 and 6, the following messages appear on the server console.

```
...
Server-select() is OK...
Server-select() is OK...
./select: socket 5 hung up
Server-select() is OK...
./select: socket 6 hung up
Server-select() is OK...
./select: socket 4 hung up
```

- There are two file descriptor sets in the code: master and read_fds. The first, master, holds all the socket descriptors that are currently connected, as well

as the socket descriptor that is listening for new connections.

- The reason we have the master set is that select() actually changes the set you pass into it to reflect which sockets are ready to read. Since we have to keep track of the connections from one call of select() to the next, we must store these safely away somewhere. At the last minute, we copy the master into the read_fds, and then call select().
- Then every time we get a new connection, we have to add it to the master set and also every time a connection closes, we have to remove it from the master set.
- Notice that we check to see when the listener socket is ready to read. When it is, it means we have a new connection pending, and we accept() it and add it to the master set. Similarly, when a client connection is ready to read, and recv() returns 0, we know that the client has closed the connection, and we must remove it from the master set.
- If the client recv() returns non-zero, though, we know some data has been received. So we get it, and then go through the master list and send that data to all the rest of the connected clients.

Connecting a TCP server and client

- The following program examples are connection-oriented where sockets use TCP to connect a server to a client, and a client to a server. This example provides more complete sockets' APIs usage.

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

```
/***********tcpserver.c***********************/
/* header files needed to use the sockets API */
/* File contain Macro, Data Type and Structure */
/*********************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <errno.h>
#include <unistd.h>
/* BufferLength is 100 bytes */
#define BufferLength 100
/* Server port number */
#define SERVPORT 3111

int main()
{
/* Variable and structure definitions. */
```

```
int sd, sd2, rc, length = sizeof(int);
int totalcnt = 0, on = 1;
char temp;
char buffer[BufferLength];
struct sockaddr_in serveraddr;
struct sockaddr_in their_addr;

fd_set read_fd;
struct timeval timeout;
timeout.tv_sec = 15;
timeout.tv_usec = 0;

/* 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("Server-socket() error");
/* Just exit */
exit (-1);
}
else
printf("Server-socket() is OK\n");

/* The setsockopt() function is used to allow */
/* the local address to be reused when the server */
/* is restarted before the required wait time */
/* expires. */
/******************************************/
/* Allow socket descriptor to be reusable */
if((rc = setsockopt(sd, SOL_SOCKET, SO_REUSEADDR,
(char *)&on, sizeof(on))) < 0)
{
perror("Server-setsockopt() error");
close(sd);
exit (-1);
}
else
printf("Server-setsockopt() is OK\n");

/* bind to an address */
```

```
memset(&serveraddr, 0x00, sizeof(struct
sockaddr_in));
serveraddr.sin_family = AF_INET;
serveraddr.sin_port = htons(SERVPORT);
serveraddr.sin_addr.s_addr = htonl(INADDR_ANY);

printf("Using %s, listening at %d\n",
inet_ntoa(serveraddr.sin_addr), SERVPORT);

/* After the socket descriptor is created, a bind()
*/
/* function gets a unique name for the socket. */
/* In this example, the user sets the */
/* s_addr to zero, which allows the system to */
/* connect to any client that used port 3005. */
if((rc = bind(sd, (struct sockaddr *)&serveraddr,
sizeof(serveraddr))) < 0)
{
perror("Server-bind() error");
/* Close the socket descriptor */
close(sd);
/* and just exit */
exit(-1);
}
else
    printf("Server-bind() is OK\n");

/* The listen() function allows the server to accept
*/
/* incoming client connections. In this example, */
/* the backlog is set to 10. This means that the */
/* system can queue up to 10 connection requests
before */
/* the system starts rejecting incoming requests.*/
/************************************************/
/* Up to 10 clients can be queued */
if((rc = listen(sd, 10)) < 0)
{
    perror("Server-listen() error");
    close(sd);
    exit (-1);
}
else
    printf("Server-Ready for client
connection...\n");

/* The server will accept a connection request */
```

```
/* with this accept() function, provided the */
/* connection request does the following: */
/* - Is part of the same address family */
/* - Uses streams sockets (TCP) */
/* - Attempts to connect to the specified port */
/*********************************************/
/* accept() the incoming connection request. */
int sin_size = sizeof(struct sockaddr_in);
if((sd2 = accept(sd, (struct sockaddr *)&their_addr,
&sin_size)) < 0)
{
    perror("Server-accept() error");
    close(sd);
    exit (-1);
}
else
    printf("Server-accept() is OK\n");

/*client IP*/
printf("Server-new socket, sd2 is OK...\n");
printf("Got connection from the f***ing client:
%s\n", inet_ntoa(their_addr.sin_addr));

/* The select() function allows the process to */
/* wait for an event to occur and to wake up */
/* the process when the event occurs. In this */
/* example, the system notifies the process */
/* only when data is available to read. */
/*********************************************/
/* Wait for up to 15 seconds on */
/* select() for data to be read. */
FD_ZERO(&read_fd);
FD_SET(sd2, &read_fd);
rc = select(sd2+1, &read_fd, NULL, NULL, &timeout);
if((rc == 1) && (FD_ISSET(sd2, &read_fd)))
{
/* Read data from the client. */
totalcnt = 0;

while(totalcnt < BufferLength)
{
/* When select() indicates that there is data */
/* available, use the read() function to read */
/* 100 bytes of the string that the */
/* client sent. */
/*********************************************/
/* read() from client */
```

```
rc = read(sd2, &buffer[totalcnt], (BufferLength -
totalcnt));
if(rc < 0)
{
    perror("Server-read() error");
    close(sd);
    close(sd2);
    exit (-1);
}
else if (rc == 0)
{
    printf("Client program has issued a close()\n");
    close(sd);
    close(sd2);
    exit(-1);
}
else
{
    totalcnt += rc;
    printf("Server-read() is OK\n");
}

}
}
else if (rc < 0)
{
    perror("Server-select() error");
    close(sd);
    close(sd2);
    exit(-1);
}
/* rc == 0 */
else
{
    printf("Server-select() timed out.\n");
    close(sd);
    close(sd2);
    exit(-1);
}

/* Shows the data */
printf("Received data from the f***ing client: %s\n",
buffer);

/* Echo some bytes of string, back */
/* to the client by using the write() */
/* function. */
```

```
/***********************************/
/* write() some bytes of string, */
/* back to the client. */
printf("Server-Echoing back to client...\n");
rc = write(sd2, buffer, totalcnt);
if(rc != totalcnt)
{
perror("Server-write() error");
/* Get the error number. */
rc = getsockopt(sd2, SOL_SOCKET, SO_ERROR, &temp,
&length);
if(rc == 0)
{
    /* Print out the asynchronously */
    /* received error. */
    errno = temp;
    perror("SO_ERROR was: ");
}
else
    printf("Server-write() is OK\n");

close(sd);
close(sd2);
exit(-1);
}

/* When the data has been sent, close() */
/* the socket descriptor that was returned */
/* from the accept() verb and close() the */
/* original socket descriptor. */
/***************************************/
/* Close the connection to the client and */
/* close the server listening socket. */
/***************************************/
close(sd2);
close(sd);
exit(0);
return 0;
}
```

- Compile and link the program. Make sure there is no error.

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

- Run the program. In this example we let the program run in the background.

```
[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...

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

- Do some verification.

```
[bodo@bakawali testsocket]$ ps aux | grep tcpserver
bodo      7914  0.0  0.2  3172  324 pts/3    S
11:59   0:00 ./tcpserver
bodo      7921  0.0  0.5  5540  648 pts/3    S+
12:01   0:00 grep tcpserver
[bodo@bakawali testsocket]$ netstat -a | grep 3111
tcp        0      0 *:3111
*:*                     LISTEN
```

- When the next program example (the TCP client) is run, the following messages should be expected at the server console.

```
[bodo@bakawali testsocket]$ 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...

[1]+  Done                    ./tcpserver
[bodo@bakawali testsocket]$
```

- If the server program and then the client are run, the following messages should be expected at the server console.

```
[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]$
```

- Just telneting the server.

```
[bodo@bakawali testsocket]$ telnet   203.106.93.94
3111
Trying 203.106.93.94...
Connected to bakawali.jmti.gov.my (203.106.93.94).
Escape character is '^]'.
^]
telnet> help
Commands may be abbreviated.  Commands are:

close           close current connection
logout          forcibly logout remote user and close
the connection
display         display operating parameters
mode            try to enter line or character mode
('mode ?' for more)
open            connect to a site
quit            exit telnet
send            transmit special characters ('send ?'
for more)
set             set operating parameters ('set ?' for
more)
unset           unset operating parameters ('unset ?'
for more)
status          print status information
toggle          toggle operating parameters ('toggle
?' for more)
slc             change state of special charaters
```

```
('slc ?' for more)
auth            turn on (off) authentication ('auth
?' for more)
encrypt         turn on (off) encryption ('encrypt ?'
for more)
forward         turn on (off) credential forwarding
('forward ?' for more)
z               suspend telnet
!               invoke a subshell
environ         change environment variables
('environ ?' for more)
?               print help information
telnet>quit
```

- Well, it looks that we have had a telnet session with the server.

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.