College of Engineering and Management, KOLAGHAT
Data Communication Laboratory (6th Semester 2021, CSE)
Experiment: TCP and UDP echo clients
(Filename - socket2.docx)

Documents to read:
1. UNIX Network Programming
By W. Richard Stevens
Chapter-6, on Berkeley Sockets

2. Manuals socket(2), ip(7),tcp(7) and udp(7)

Square brackets are used to separate keywords, such as filenames, programs names etc etc from lines of text. They have no other meaning.

T-2 $ mkdir $HOME/socket2
T-2 $ cd $HOME/socket2

A few internal servers (echo, daytime, time etc etc) run on a host, using TCP and UDP protocols. In this experiment, the two internal [echo] servers (using TCP and UDP) will be used, to test the TCP and UDP [echo] client programs.

Port number seven is assigned to [echo] servers. The following command and part of the possible output is shown...
T-2 $ cat /etc/services | grep echo
echo 7/tcp
echo 7/udp

However the [echo] servers can be disabled by commenting appropriate lines in file [/etc/inetd.conf].

Use a port scanner to check port-7 of your host

nmap -sTU 127.0.0.1 -p 7

If the two [echo] servers (using TCP and UDP) are not enabled in your host, they are to be enabled by editing file [/etc/inetd.conf]and sending SIGHUP to [inetd] process.

The TCP [echo] server can be checked using [telnet] client

$ telnet 127.0.0.1 7
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
Are you the echo server?

Are you the echo server?

End session with CTRL-] and then [q].

The UDP [echo] server cannot be checked as yet from the command line.

TCP [echo] client:
The system calls to be executed by the client and server, for connection oriented protocol, are repeated below, from [socket1.txt]. TCP is a connection oriented protocol.

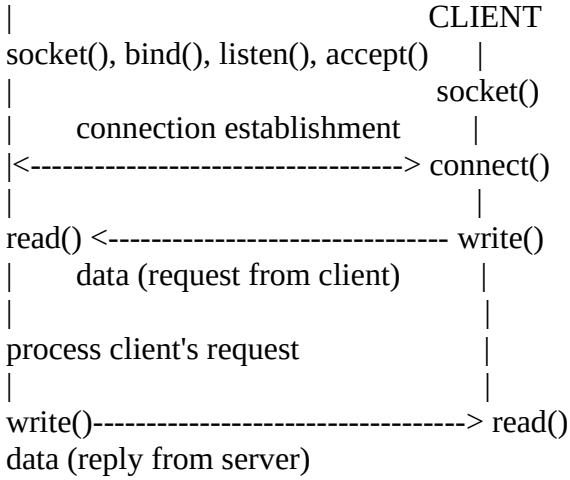

Figure-1

The program [f1.c] is the TCP [echo] client. This program has to call [socket] and [connect] calls to establish connection with a server. Then the client sends request to server and reads reply from server.

In [f1.c] the IP address of the [echo] server and TCP port of the [echo] server are supplied as arguments to the program. We know that assigned number of [echo] server is 7. Still we are using port number as an argument to the client program. This would enable us to connect to experimental [echo] servers using any free port number.

Save the following file as [f1.c]

```
//file-name f1.c   TCP echo client
// usage -> program-name server-address  server-port

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

```
#include <stdio.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <stdlib.h>


#define BUFFERSIZE     1024   // arbitrary

//#define CHECK

int main(int argc, char *argv[])
{
int                sd;  // used as socket descriptor
ssize_t              i,j;
int                n; // used to check returned value
char                buffer[BUFFERSIZE];
struct  sockaddr_in   server_addr;    // An IPv4 address

if( argc != 3 )
{
printf("Usage:   %s server-address  server-port \n", argv[0] );
printf("Example: %s 192.168.5.148   12345 \n", argv[0] );
exit(1);
}

printf( "TCP-echo-client starting...\n" );

// A  TCP ( IPv4 stream )  socket  is created.
sd = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP );
if( sd == -1 ) { perror("socket-call"); exit( 1 ); }

/*
The descriptor [sd] is not useful now. This is to be connected to server's address with [connect() ]
system call.  Before making this call, the  members of  structure [ server_addr ] are to  be initialized.

An IP  socket  address  is  defined as a  combination  of  an IP interface address and a port number,
using a protocol.

IPv4 address is a structure [ struct sockaddr_in ]

The structure has the following members:

struct sockaddr_in {
sa_family_t          sin_family;                // address family: AF_INET
u_int16_t            sin_port;                  // port in network byte order
struct       in_addr   sin_addr;                // Internet address
};
```

Internet address:

```
struct in_addr {
u_int32_t      s_addr;                          // address in network byte order
};
```

You may read manual [ip(7)].

The variable [server_addr], used by you, is an instance of the structure [struct sockaddr_in]

*/

// First member of structure [server_addr] is initialized with the address family consistent with the //protocol family of the socket.

```
server_addr.sin_family = AF_INET;
```

/*

The port number is to be converted from host byte order to network byte order. Network byte order is big endian, whereas host byte order may be little endian or big endian. In your PC (i80x86) host byte order is little endian. The function [htons()] is used to convert port number from host byte order to network byte order. Even if the host byte order is big endian, calling [htons()] is not a mistake. You may read manual [htons(3)].

argv[2] is the port number in our program.

*/

// Second member of structure [server_addr] is initialised

```
server_addr.sin_port = htons( atoi(argv[2]) );
```

/*

Internet host address given as standard numbers-and-dots notation is to be converted into binary data and is to be stored as the [server_addr.sin_addr] of [server_addr] structure. The function [inet_aton()] is used to do that. This function returns nonzero if the address is valid, zero if not. You may read manual [inet_aton(3)].

*/

// Third member of structure [server_addr] is initialised.

// argv[1] is the server address in dotted decimal quad

```
n = inet_aton( argv[1], &(server_addr.sin_addr) );
if( n == 0 ) { printf("inet_aton-Invalid address\n"); exit(1); }


n = connect( sd, (struct sockaddr *) &server_addr, sizeof(server_addr) );
if( n == -1 ) { perror("connect-call"); exit(1); }
```

// the descriptor [sd] is now connected to server's socket

```
#ifdef CHECK
printf("Check with [ # netstat -tn ] in another terminal\n" );
printf("Check TCP connection establishment\n" );
while(1) { sleep(1); }  // endless loop
```

```
#endif

/*
[bind()] system call was not called on [sd] before [ connect() ] system call. As [connect()] was called on a unbound socket, [sd] was automatically bound to a random free port ( ephemeral port ) with the local address set to [INADDR_ANY].
*/

write( STDOUT_FILENO, "Enter the string:", 17 );

// clear buffer before reading
memset( buffer, '\0', BUFFERSIZE );
i = read( STDIN_FILENO, buffer, BUFFERSIZE );
if( i == -1 ) { perror("read1"); exit(1); }
printf( "bytes read from keyboard=%u\n", i );

// write contents of buffer on server's socket
j = write( sd, buffer, i );
if( j == -1 ) { perror("write1"); exit(1); }
printf("bytes written in server's socket=%u\n",j);

// clear buffer before reading
memset( buffer, '\0', BUFFERSIZE );
// read from server's socket into buffer
i = read( sd, buffer, BUFFERSIZE );
if( i == -1 ) { perror("read2"); exit(1); }
printf("bytes read from server's socket=%u\n", i );

write( STDOUT_FILENO,"Reply from echo server->", 24 );
j = write( STDOUT_FILENO, buffer, i );
if( j == -1 ) { perror("write2"); exit(1); }

// Shutdown the both-way ( duplex ) connection.
shutdown(sd, SHUT_RDWR);

return 0;
}
```

Assignment-1:

(a) Define CHECK in the program [f1.c]. Compile and execute.....

```
$ gcc -Wall ./f1.c -o ./one-a
```

Run the program using wrong number of arguments to see usage message

```
$ ./one-a
$ ./one-a 192.168.5.251
```

Run the program correctly. Replace 192.168.5.251 with a suitable server address
$ ./one-a 192.168.5.251 7

From another terminal use [# netstat -tnp] command. From the output of the command, record the line related to your program

Proto Local Address Foreign Address State PID/Program name

Terminate with CTRL-C.

Try to connect to a non-existent TCP server in localhost or in another host. The [connect] call of your program should fail with a suitable error message.

$ ./one-a 127.0.0.1 2000
$ ./one-a 192.168.5.251 7777

(b) Undefine(comment) CHECK in [f1.c] and compile
$ gcc -Wall ./f1.c -o ./one-b

Connect to TCP [echo] server listening on loopback interface
$ ./one-b 127.0.0.1 7

Enter a string to the server. It should be echoed on your terminal.

Connect to TCP [echo] server listening on eth0 interface of your host
$ ./one-b IP-address-of-eth0-interface 7

Try to connect to TCP [echo] servers running on different hosts in the laboratory or outside the laboratory. Your TCP [echo] client program should work if TCP [echo] servers are enabled in the specified hosts.

$ ./one-b 127.0.0.1 7
$ ./one-b 192.168.5.251 7
$ ./one-b 192.168.5.252 7
$ ./one-b 192.168.5.160 7
$ ./one-b 172.16.1.100 7
$ ./one-b 210.212.4.3 7

Rename the executable [one-b] as [tcp-echo-client].

$ mv ./one-b ./tcp-echo-client

Create a directory [bin] under your home directory
$ mkdir $HOME/bin

Copy the executable in [bin] directory
$ cp ./tcp-echo-client $HOME/bin/

Include the [bin] directory in [PATH] environmental variable

$ PATH=$PATH:$HOME/bin/

Check the intended inclusion with...
$ echo $PATH

If the inclusion is correct, you should be able to run the TCP client from any directory of your account. If you intend to include the [bin] directory in PATH variable, on every login, include the following line at the end of file [.bash_profile], in your home directory.

```
#--  .bash_profile  file  --
.
.
.
PATH=$PATH:$HOME/bin/
#--------------------------
```

The executable [tcp-echo-client] should not be deleted. This would be used to connect to TCP [echo] server, to be written by you.

End of Assignment-1.

The next program [f2.c] is a TCP [echo] client which connects to port-7 of standard TCP [echo] servers. [f1.c] is modified as [f2.c]. The port number is no longer supplied as an argument.

```
------------------------------------------------------------------
//file-name f2.c   TCP echo client
// usage -> program-name server-address

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


#define BUFFERSIZE     1024
```

```
#define PORT          7


int main(int argc, char *argv[])
{
int                 sd,n;
ssize_t              i,j;
char                buffer[BUFFERSIZE];
struct  sockaddr_in   server_addr;

if( argc != 2 )
{
printf("Usage: %s server-address \n", argv[0] );
exit(1);
}


sd = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP );
if( sd == -1 ) { perror("socket-call"); exit( 1 ); }

server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( PORT );
n = inet_aton( argv[1], &(server_addr.sin_addr) );
if( n == 0 ) { printf("inet_aton-Invalid address\n"); exit(1); }

n = connect( sd, (struct sockaddr *) &server_addr,
sizeof(server_addr) );
if( n == -1 ) { perror("connect-call"); exit(1); }

write( STDOUT_FILENO, "Enter the string:", 17 );

memset( buffer, '\0', BUFFERSIZE );
i = read( STDIN_FILENO, buffer, BUFFERSIZE );
if( i == -1 ) { perror("read1"); exit(1); }
printf( "bytes read from keyboard=%u\n", i );

j = write( sd, buffer, i );
if( j == -1 ) { perror("write1"); exit(1); }
printf("bytes written in server's socket=%u\n",j);

memset( buffer, '\0', BUFFERSIZE );
i = read( sd, buffer, BUFFERSIZE );
if( i == -1 ) { perror("read2"); exit(1); }
printf("bytes read from server's socket=%u\n", i );

write( STDOUT_FILENO,"Reply from echo server->", 24 );
j = write( STDOUT_FILENO, buffer, i );
if( j == -1 ) { perror("write2"); exit(1); }
```

```
shutdown(sd, SHUT_RDWR);

return 0;
}
```

Assignment-2:

Compile [f2.c]
$ gcc -Wall ./f2.c -o ./two

Connect to some standard TCP [echo] servers
$ ./two 127.0.0.1
$ ./two 192.168.5.146

End of Assignment-2.

UDP [echo] client:

The system calls using a connection-less protocol is shown below.
(From "UNIX Network Programming" by W.Richard Stevens).

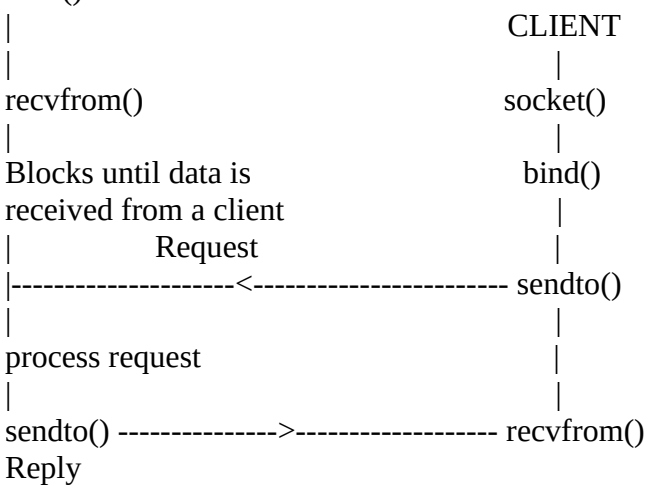

Figure-2

[f3.c] is the UDP client program. From the above figure it is observed that [f3.c] has to call [socket()], [bind()], [sendto()] and [recvfrom()] calls.

In connection-less communication, data are put into packets and each packet is an independent communication. UDP packets may arrive at the destination in wrong order or not at all. If the packet is lost or there is no process to accept the data at the destination address, usually the sending end system does not get an indication.

Note: If there is no data present, [recvfrom()] call would block.

[listen()] and [accept()] calls are not allowed in connection-less communication.
[connect()] function may be used on a datagram socket. This would specify a destination to send further packets. In such cases [send()] function can be used to send data on a datagram socket.

Sending datagrams:

Normally [sendto()] function is used to send data on a datagram socket.

[sendto()] call:

int sendto(socket-fd, buffer, buffer-size, flag, dest-address, dest-address-length)

Number of bytes specified by [buffer-size] in [buffer] are transmitted through the socket, specified by [socket-fd], to the destination address specified by [dest-address] having a length [dest-address-length].

The [flag] argument is a bitwise-OR of the following:

MSG_OOB
MSG_DONTROUTE

The flag argument may be zero if none of the above are used. For the correct data types of the arguments, read manual [sendto(2)].

Receiving datagrams:
Normally [recvfrom()] call is used to receive one datagram from a socket. This system call also stores the address from which the datagram came.

[recvfrom()] call:

int recvfrom(socket-fd, buffer, buffer-size, flag, src-address, src-address-length)

This call reads one datagram from the socket specified by[socket-fd], in the buffer specified by [buffer]. The maximum number of bytes to be read are specified by [buffer-size].

The [src-address] and [src-address-length] specify the address, the datagram came from.

For full description and data types of the arguments, read the manual of [recvfrom(2)].

A null pointer can be used as [src-address] and zero as [src-address-length] if this information is not needed.[recv()] call may be used in such situation.

The [read()] call may be used if [flag] argument is not essential.

In the following example a datagram is read in four different ways:

[recvfrom()] in full form,
[recvfrom()] in stripped form,
[recv()] and
[read()].

In [f3.c], server address and server port number, are supplied as two arguments to the program.

```
------------------------------------------------------------------------
// file-name f3.c   UDP echo client
// usage -> program-name server-address  server-port

#include    <sys/socket.h>
#include    <stdio.h>
#include    <arpa/inet.h>
#include    <unistd.h>
#include    <stdlib.h>

#define     SIZE    512

// Uncomment the following line to use [recvfrom()] call in full form.
//#define   RECVFROM_FULL

// Uncomment the following line  to use [recvfrom()] call in  stripped form
//#define   RECVFROM_STRIPPED

// Uncomment the following line to use [recv()] call
//#define   RECV

// Uncomment the following line to use [read()] call
//#define   READ

// Uncomment the following line to print numeric values of some symbolic constants
//#define   DEBUG

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

int  sd; //  socket descriptor

int  n;    // to check returned value of function calls.
```

```
ssize_t              i;
char                 buffer[SIZE];
struct   sockaddr_in   server_addr;
struct   sockaddr_in   source_addr;


#ifdef RECVFROM_FULL
// following variables are used to illustrate use of [ recvfrom ] call in full form
socklen_t            source_addr_length;
unsigned long          nbo;  // host address in network byte order
unsigned long          hbo;  // host address in network byte order
char                 *host_addr;
#endif

if( argc != 3 )
{
printf("usage -> prog-name  server-address  server-port \n");
printf("example -> %s 172.16.2.15   7 \n", argv[0] );
exit(1);
}

printf("udp-echo-client: PID of the program = %u\n", getpid() );

// an IPv4 datagram socket ( UDP socket ) is created
sd = socket( AF_INET, SOCK_DGRAM, IPPROTO_UDP );
// zero may be used for IPPROTO_UDP
if( sd == -1 )  { perror("socket-call"); exit(1); }

// Initialise the members of server address
server_addr.sin_family = AF_INET;
n = inet_aton( argv[1], &(server_addr.sin_addr) );
if( n == 0 ) { perror("invalid-address"); exit(1); }
server_addr.sin_port = htons( atoi(argv[2]) );


write( STDOUT_FILENO, "u-e-c: Enter the string:", 24 );
memset( buffer, '\0', SIZE );
i = read( STDIN_FILENO, buffer, SIZE );

n = sendto( sd, buffer, i, 0, ( struct sockaddr *) &server_addr, sizeof( server_addr) );
if( n == -1 ) { perror("sendto-call"); exit(1); }

printf("u-e-c: Sent %u bytes to server \n", n );

// clear the buffer, before receiving from server
memset( buffer, '\0', SIZE );
```

```
#ifdef RECVFROM_FULL
printf("u-e-c: Using recvfrom() call in full form \n");
source_addr_length = sizeof(source_addr);
n = recvfrom( sd, buffer,SIZE, MSG_PEEK, (struct sockaddr *) &source_addr, &source_addr_length
);
if( n == -1 ) { perror("recvfrom-call-full"); exit(1); }

// Print the value of a symbolic constant we are interested in
printf("recvfrom-full: AF_INET = %u \n", AF_INET);

/* [recvfrom()] system call has filled in members of the variable [source_addr]. The values of the
members are now retrieved. */

printf("rf-full: source address family = %u\n", source_addr.sin_family );

nbo = source_addr.sin_addr.s_addr;
printf("rf-full: source address in network byte order ");
printf("= %lX Hex \n",  nbo );

hbo = ntohl(source_addr.sin_addr.s_addr);
printf("rf-full: source address in host byte order = %lX Hex \n", hbo );

host_addr = inet_ntoa(source_addr.sin_addr);

printf("rf-full: source address in dotted decimal quad = %s  \n", host_addr );

printf("rf-full: source port in network byte order = %X Hex \n", source_addr.sin_port );

printf("rf-full: source port in host byte order = %X Hex\n", ntohs(source_addr.sin_port) );

printf("rf-full: source address length = %u bytes\n", source_addr_length );
#endif


#ifdef RECVFROM_STRIPPED
printf("u-e-c: Using recvfrom() call in stripped form \n");
//We are not interested in source address
n = recvfrom( sd, buffer,SIZE, 0, NULL, 0 );
if( n == -1 ) { perror("recvfrom-call-stripped"); exit(1); }
#endif

#ifdef RECV
printf("u-e-c: Using recv() call \n");
// We are not interested in source address but want to use [flag]
n = recv( sd, buffer, SIZE, MSG_PEEK );
if( n == -1 ) { perror("recvfrom-call-stripped"); exit(1); }
#endif
```

```
#ifdef READ
printf("u-e-c: Using read() call \n");
// We  are not interested in source address and don't want to use  [flag]
n = read( sd, buffer, SIZE);
if( n == -1 ) { perror("read-call"); exit(1); }
#endif


printf("u-e-c: Received %u bytes from server \n", n );
write( STDOUT_FILENO, "u-e-c: From server->", 20 );
write( STDOUT_FILENO, buffer, n );

shutdown( sd, SHUT_RDWR );

return 0;
}
```

Assignment-3:

(a) Define only READ and compile
$ gcc -Wall ./f3.c -o ./three-a

Connect to any UDP [echo] server
$ ./three-a 192.168.5.251 7

Did the program run correctly? (Y / n)

(b) Undefine READ, define only RECV and compile
$ gcc -Wall ./f3.c -o ./three-b

Connect to any UDP [echo] server
$ ./three-b 192.168.5.149 7

Did the program run correctly? (Y / n)

(c) Undefine RECV, define only RECVFROM_STRIPPED and compile
$ gcc -Wall ./f3.c -o ./three-c

Connect to any UDP [echo] server
$ ./three-c 192.168.5.161 7

Did the program run correctly ? (Y / n)

(d) Comment RECVFROM_STRIPPED, define only RECVFROM_FULL and compile
$ gcc -Wall ./f3.c -o ./three-d

Connect to any [echo] server using UDP
$ ./three-d ___.___.___.___ 7

Did the program run correctly? (Y / n)

Record the following...
Value of symbolic constant AF_INET =
source protocol family =
source address in network byte order = Hex
source address in host byte order = Hex
source address in dotted decimal quad =
source port in network byte order = Hex
source port in host byte order = Hex
source address length = bytes

End of Assignment-3.