College of Engineering and Management, KOLAGHAT

Data Communication Laboratory (6th Semester 2021, CSE)

Experiment: ECHO-servers, UDP and TCP. (Filename - socket3.txt)

Create a directory [socket3] in your home directory.

T-2 $ mkdir $HOME/socket3

T-2 $ cd $HOME/socket3

In Unix machines, [echo] servers, using TCP and UDP are usually provided. Port number seven is assigned to [echo] servers. In this experiment you are going to write and test ECHO severs, using some non-standard ports.

The following figure shows socket system call for connection oriented protocol. This figure is repeated from a previous experiment. TCP is a connection oriented protocol.

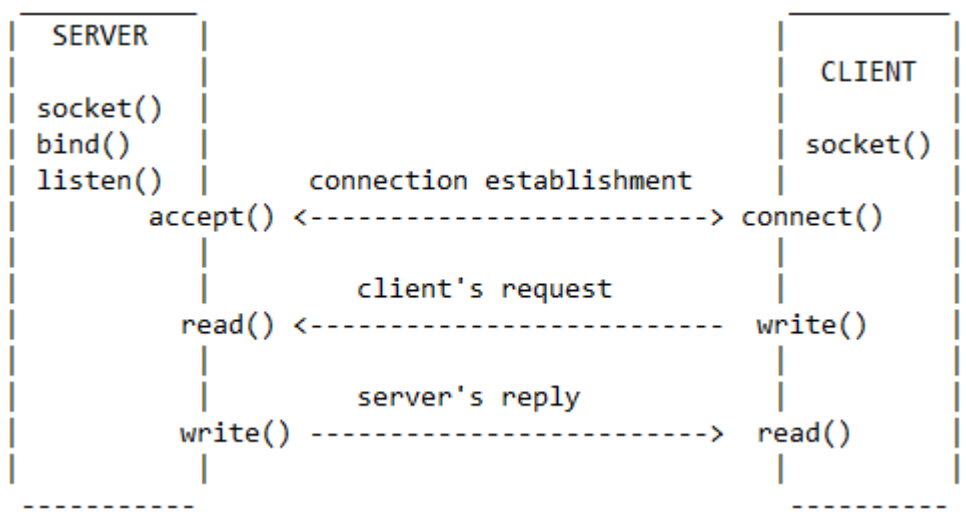


Figure-1

A stand alone ECHO server using TCP protocol, is to be written by you. It can be seen from Figure-1, that the server has to

create a IPv4 stream (TCP) socket,

bind the socket to local address / addresses,

listen on the socket for incoming requests and

wait to accept client's requests on the socket.

The returned value of [accept()] call creates a new socket. This new socket is used for communication with the client. Save the following as [f1a.c]. This program has limitations.

```
// file-name f1a.c

// TCP ECHO server listening on only [ eth0 ] interface

// usage: program-name


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

#define   ETH0        "address-of-your-eth0-interface"
// next line is an example
// #define   ETH0     "192.168.5.167"

#define   SERVER_PORT  50001     // an unused port in your HOST
/* Choose any port number between 50000 and 50030.  This range of ports has no special
significance.  This range of ports are scanned in this experiment to check on servers running
on different hosts.*/

#define   BUFFERSIZE    1024  // arbitrary

int main( int argc, char *argv[] )
{
  int                ser_sd;
  // [ser_sd] would be used as server's socket file descriptor

  int                tempsockfd, n ;
  //int              set_it=1;
  socklen_t            clientlength;
  ssize_t             i, j ;
  char                buffer[BUFFERSIZE];
  struct  sockaddr_in   server_addr, client_addr;


  // the program takes no argument
  if( argc != 1 )
  {
    printf("usage: %s \n", argv[0] );
    exit(1);
  }


  // a TCP socket( IPv4 stream socket ) is created
  ser_sd = socket( PF_INET, SOCK_STREAM, IPPROTO_TCP );
  if( ser_sd == -1 ) { perror("socket-call"); exit(1); }
  // zero may be used for IPPROTO_TCP


  /*
  // Use this call to reuse address.
  // This call has 5 arguments
  n = setsockopt(  ser_sd,
                SOL_SOCKET,
```

```
            SO_REUSEADDR,
            &set_it,
            sizeof(int)
          );
if( n == -1 ) { perror("setsockopt"); exit(1); }
*/


// Initialise the members of the local address (server's address)

// address-family of server's address
server_addr.sin_family = AF_INET;

// local-address ( server's IP address )
n = inet_aton( ETH0, &(server_addr.sin_addr) );
if( n == 0 ) { printf("inet_aton: Invalid address\n"); exit(1); }

// local-port ( server's port )
server_addr.sin_port = htons(SERVER_PORT);

// bind the server's address with the created socket.
// [bind] call has three arguments
n = bind( ser_sd, (struct sockaddr *) & server_addr, sizeof(server_addr) );
if( n == -1) { perror("bind-call"); exit(1); }


// putting the socket in listening state
n = listen( ser_sd, 1 );
if( n == -1 ) { perror("listen-call"); exit(1); }

clientlength = sizeof(client_addr);
// [clientlength] will be used as third argument of [accept] call

while ( 1 ) // endless loop
{
  printf( "%s : waiting for client's request on port %u \n", argv[0], SERVER_PORT );

  // [accept()] call has three arguments
  tempsockfd = accept( ser_sd, (struct sockaddr *)&client_addr, &clientlength );
  if( tempsockfd == -1 ) { perror("accept-call"); }
  // [tempsockfd] is used for communication with the current client


  // clear buffer before reading into it
  memset( buffer, '\0', BUFFERSIZE );
  // read client's request in the buffer
  i = read( tempsockfd, buffer, BUFFERSIZE );
  if( i == -1 ) { perror("socket-read"); exit(1); }

  j = write( STDOUT_FILENO,"Received from client->", 22 );
  if( j == -1 ) { perror("screen-write"); exit(1); }

  printf("%s \n", buffer );

  // write contents of buffer to client's socket
```

```
    j = write( tempsockfd, buffer, i );
    if( j == -1 ) { perror("socket-write"); exit(1); }

    // shutdown communication channel with this client
    n = shutdown(tempsockfd,SHUT_RDWR);
    if ( n == -1 ) { perror("shutdown"); exit(1); }

    // remove this useless socket
    close(tempsockfd);
  } // endless-loop block

  // program would not reach here due to endless-loop

  // shutdown(ser_sd,SHUT_RDWR);
  // close(ser_sd);
  // exit(0);
} // end of main
```

In the above program the server's IP address is hard coded. If this program is compiled in another host, that host's IP address is to be used in the program. This is restrictive, as the program would not run correctly in another host, without modification. So this program is not proper. The limitations of this program are to be examined.

Assignment-1:

Define the ETH0 correctly, compile [f1a.c] and run the executable
T-2 $ gcc -Wall ./f1a.c -o ./one-a
T-2 $ ./one-a

Server listening on port _____

Verify the server's port with
T-5 # netstat -atnp | grep one-a
Was the server listening on eth0 interface? (Y / n)
Was the server listening on loopback interface? (y / N)

Terminate the server with CTRL-C
T-2 CTRL-C

Login in your account in terminal four and test the server. Use the program [tcp-echo-client], written by you in a previous experiment.
T-4 $ tcp-echo-client 192.168.5.___ port-number

Did the [echo] server run correctly? (Y / n)

Request your friend to connect to your TCP [echo] server
friend $ tcp-echo-client your-ip-address your-port-number

Request your friend to enter a string.

Was your friend able to communicate with your [echo] server? (Y / n)

This is an iterative server. This server can process a single client's request at a time.

Start the server
T-2 $ ./one-a

From Terminal-3, run the client program
T-3 $ tcp-echo-client 192.168.5.___ _____

Do not enter a string

From Terminal-4, run the client program
T-4 $ tcp-echo-client 192.168.5.___ _____
Enter an input string.

Did you get the echo? (y / N)

Enter an input string in terminal-3

Did you get the echo? (Y / n)

Go back to terminal-4

Did you get the echo? (Y / n)

Terminate server with CTRL-C.

Use any port under 1024, as the server's port. You might define the port in [f1a.c], as shown below

#define SERVER_PORT 1023

Save the modified file as [f1b.c].

Compile...
T-2 $ gcc -Wall ./f1b.c -o ./one-b

Try to start the server as an ordinary user.
T-2 $ ./one-b

Was an ordinary user able to start the server ? (y / N)

Start the server as [root] user, from [socket3] directory
T-2 $ su
enter [root] password
T-2 # ./one-b

Did the server start ? (Y / n)

From another terminal use [tcp-echo-client] to test the server
T-3 $ tcp-echo-client 127.0.0.1 server's-port

Did the server run correctly? (Y / n)

Terminate server with CTRL-C.
T-2 CTRL-C

```
T-2 # exit
T-2 $
```

End of Assignment-1.

In the previous programs the variable [client_addr] was used as an argument of [accept] call. [accept] call may not use this variable. In [f2.c], a short form of [accept] call is used. In short form of the [accept] call, the variable [clientlength] is also not needed.

```
// file-name f2.c
// TCP ECHO server listening on [ eth0 ] interface.
// short form of [accept] call is used.
// usage: program-name


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

#define   SERVER_ADDRESS     "eth0-interface-address"

#define   SERVER_PORT        50000

#define   BUFFERSIZE         1024


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


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

  server_addr.sin_family = AF_INET;
  n = inet_aton( SERVER_ADDRESS, &(server_addr.sin_addr) );
  if( n == 0 ) { printf("Invalid address\n"); exit(1); }
  server_addr.sin_port = htons(SERVER_PORT);

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

  n = listen( ser_sd, 1 );
  if( n == -1 ) { perror("listen-call"); exit(1); }

  while ( 1 ) // endless loop
  {
```

```
   printf( "%s : waiting for client's request on port %u \n", argv[0], SERVER_PORT );

   // note the 2nd and 3rd arguments
   tempsockfd = accept( ser_sd, NULL, 0 );
   if( tempsockfd == -1 ) perror("accept-call");

   memset( buffer, '\0', BUFFERSIZE );
   i = read( tempsockfd, buffer, BUFFERSIZE );
   if( i == -1 )
   {
     perror("socket-read");
     exit(1);
   }

   j = write( STDOUT_FILENO,"Received from client->", 22 );
   if( j == -1 )  perror("screen-write");

   printf("%s \n", buffer );

   j = write( tempsockfd, buffer, i );
   if( j == -1 )
   {
      perror("socket-write");
      exit(1);
   }

   n = shutdown(tempsockfd,SHUT_RDWR);
   if ( n == -1 )
   {
      perror("shutdown");
      exit(1);
   }

   close(tempsockfd);
  } // end of while block

} // end of main
```

Assignment-2:

Define server's address correctly, compile and start the server
T-2 $ gcc -Wall ./f2.c -o ./two
T-2 $ ./two

From another terminal test the server
T-3 $ tcp-echo-client 192.168.5.___ 50000

Did the server run correctly ? (Y / n)

End of Assignment-2.

When full form of the [accept] call is used, the members of [client_addr] structure are filled in after a successful [accept] call. The values of a client's address and port number may be printed from the server. However this is not essential for server operation.

```
/ file-name f3.c
// TCP ECHO server listening on [ eth0 ] interface.
// prints IP address and port-number of each client
// usage: program-name


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

#define   SERVER_ADDR   "eth0-interface-address"

#define   SERVER_PORT   50000

#define   BUFFERSIZE    1024


int main( int argc, char *argv[] )
{
  int               ser_sd;
  int               tempsockfd, n ;
  socklen_t           clientlength;
  ssize_t             i, j ;
  char              buffer[BUFFERSIZE];
  struct  sockaddr_in   server_addr, client_addr;


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

  server_addr.sin_family = AF_INET;
  n = inet_aton( SERVER_ADDR, &(server_addr.sin_addr) );
  if( n == 0 ) { printf("inet_aton: Invalid address\n"); exit(1); }
  server_addr.sin_port = htons(SERVER_PORT);

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

  n = listen( ser_sd, 1 );
  if( n == -1 ) { perror("listen-call"); exit(1); }

  clientlength = sizeof(client_addr);

  while ( 1 ) // endless loop
  {
    printf( "%s : waiting for client's request on port %u \n",argv[0],SERVER_PORT );

  tempsockfd = accept( ser_sd, (struct sockaddr *) &client_addr, &clientlength );
```

```
   if( tempsockfd == -1 ) perror("accept-call");

   // IP address and port of a client is printed
   printf("client's port = %u \n",  ntohs( client_addr.sin_port ) );
   printf("client's address = %s \n", inet_ntoa(client_addr.sin_addr) );


   memset( buffer, '\0', BUFFERSIZE );
   i = read( tempsockfd, buffer, BUFFERSIZE );
   if( i == -1 )
   {
     perror("socket-read");
     exit(1);
   }

   j = write( STDOUT_FILENO,"Received from client->", 22 );
   if( j == -1 )  perror("screen-write");

   printf("%s \n", buffer );

   j = write( tempsockfd, buffer, i );
   if( j == -1 )
   {
      perror("socket-write");
      exit(1);
   }

   n = shutdown(tempsockfd,SHUT_RDWR);
   if ( n == -1 )
   {
      perror("shutdown");
      exit(1);
   }

   close(tempsockfd);
  }

} // end of main
```

Assignment-3:

Define server's address correctly, compile and run the server
T-2 $ gcc -Wall ./f3.c -o ./three
T-2 $ ./three

Request a friend (IP-address =) to test your sever

friend $ tcp-echo-client 192.168.5.___ _____

Before entering a string, friend finds out his ephemeral port for this connection.

friend in another terminal T-5 # netstat -atn

friend's ephemeral port -->

Record from your server's output

client's port--------->
client's address------>

Terminate server with CTRL-C
T-2 CTRL-C
T-2 $

End of Assignment-3.

Server's IP address was hard coded in previous servers. Usually a server listens on all available interfaces of a host. This is shown in program [f4.c]. SERVER_ADDRESS is not defined in this program.

```
// file-name f4.c
// TCP ECHO server listening on all interfaces.
// prints client's IP address and port-number
// usage: program-name


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

#define   SERVER_PORT   50000

#define   BUFFERSIZE    1024


int main( int argc, char *argv[] )
{
  int               ser_sd;
  int               tempsockfd, n ;
  socklen_t           clientlength;
  ssize_t             i, j ;
  char              buffer[BUFFERSIZE];
  struct  sockaddr_in   server_addr, client_addr;

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

  server_addr.sin_family = AF_INET;

  /*
    If INADDR_ANY is specified in the bind call the socket  will be bound to all local
interfaces.
  */
  server_addr.sin_addr.s_addr = htonl( INADDR_ANY );

  server_addr.sin_port = htons(SERVER_PORT);
```

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

 n = listen( ser_sd, 1 );
 if( n == -1 ) { perror("listen-call"); exit(1); }

 clientlength = sizeof(client_addr);

 while ( 1 ) // endless loop
 {
  printf( "%s : waiting for client's request on port %u \n", argv[0], SERVER_PORT );

  tempsockfd = accept( ser_sd, (struct sockaddr *) &client_addr, &clientlength );
  if( tempsockfd == -1 ) perror("accept-call");

  // IP address and port of a client is printed
  printf("client's port = %u \n", ntohs( client_addr.sin_port ) );
  printf("client's = %s \n", inet_ntoa(client_addr.sin_addr) );


  // clear buffer before reading into it
  memset( buffer, '\0', BUFFERSIZE );
  i = read( tempsockfd, buffer, BUFFERSIZE );
  if( i == -1 )
  {
   perror("socket-read");
   exit(1);
  }

  j = write( STDOUT_FILENO,"Received from client->", 22 );
  if( j == -1 )  perror("screen-write");

  printf("%s \n", buffer );

  j = write( tempsockfd, buffer, i );
  if( j == -1 )
  {
    perror("socket-write");
    exit(1);
  }

  n = shutdown(tempsockfd,SHUT_RDWR);
  if ( n == -1 )
  {
    perror("shutdown");
    exit(1);
  }

  close(tempsockfd);
 }

} // end of main
```

Assignment-4:

Compile and run the server
T-2 $ gcc -Wall ./f4.c -o ./four
T-2 $ ./four

From another terminal test the server using [eth0] interface
T-3 $ tcp-echo-client 192.168.5.___ _____

Did the server run correctly ? (Y / n)

From another terminal test the server using loopback interface
T-3 $ tcp-echo-client 127.0.0.1 _____

Did the server run correctly ? (Y / n)

Stop server with CTRL-C and restart it
T-2 $ ./four

Use the following command and record output.
T-5 # netstat -atnp | grep four

Proto Local Address Foreign Address State PID/Program name
tcp LISTEN _____/four

From another terminal connect to the server
T-3 $ tcp-echo-client 192.168.5.____ ______

Do not enter any string now...

Use [netstat] command and record output. A typical output is shown
T-5 # netstat -atnp | grep four

Proto Local Address Foreign Address State PID/name

tcp 0 0.0.0.0:_____ 0.0.0.0:* LISTEN ____/four
tcp 192.168.5.___:____ 192.168.5.___:____ ESTABLISHED ____/four

Enter a string in the client program.

Test the server again
T-3 $ tcp-echo-client 127.0.0.1 ______

Do not enter any string now...

Use [netstat] command and record output. A typical output is shown
T-5 # netstat -atnp | grep four

Proto Local Address Foreign Address State PID/name
tcp 0 0.0.0.0:_____ 0.0.0.0:* LISTEN four
tcp 127.0.0.1:_____ 127.0.0.1:_____ ESTABLISHED four

Run a TCP port scan on [eth0] interface for the server's port

End of Assignment-4.

[f5.c] is a concurrent TCP ECHO server.

```
// file-name f5.c
// TCP ECHO concurrent server.
// usage: program-name

#include  <sys/socket.h>
#include  <netinet/in.h>
#include  <stdio.h>
#include  <arpa/inet.h>
#include  <unistd.h>
#include  <signal.h>
#include  <wait.h>

#define   SERVER_PORT   50000

#define   BUFFERSIZE    1024

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

  act.sa_handler = pipe_handler;
  sigaction( SIGPIPE, &act, NULL );

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

  server_addr.sin_family = AF_INET;
  server_addr.sin_addr.s_addr = htonl( INADDR_ANY );
  server_addr.sin_port = htons(SERVER_PORT);

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

  n = listen( ser_sd, 1 );
  if( n == -1 ) { perror("listen-call"); exit(1); }

  while ( 1 ) // endless loop
  {
    pid_t   pid, p;

    printf( "%s : waiting for client's request on port %u \n", argv[0], SERVER_PORT );

    tempsockfd = accept( ser_sd, NULL, 0 );
    if( tempsockfd == -1 ) perror("accept-call");

    // server( main ) forks a child to handle a client's request
```

```
    pid = fork();
    if ( pid == -1 ) { perror("fork-call"); exit(1); }

    if ( pid == 0 ) // child process handles a client's request
    {
      // clear buffer before reading into it
      memset( buffer, '\0', BUFFERSIZE );
      i = read( tempsockfd, buffer, BUFFERSIZE );
      if( i == -1 )
      {
        perror("socket-read");
        exit(1);
      }

      j = write( STDOUT_FILENO,"Received from client->", 22 );
      if( j == -1 )  perror("screen-write");

      printf("%s \n", buffer );

      j = write( tempsockfd, buffer, i );
      if( j == -1 )
      {
       perror("socket-write");
       exit(1);
      }

      n = shutdown(tempsockfd,SHUT_RDWR);
      if ( n == -1 )
      {
       perror("shutdown");
       exit(1);
      }

      close(tempsockfd);
      exit(0);
    } // end of child's block

    // rest is parent's block. Main closes useless socket  [tempsockfd]
    close(tempsockfd);


  } // end of endless-loop block

} // end of main
```

Assignment-5:
Compile and run the server
T-2 $ gcc -Wall ./f5.c -o ./five
T-2 $ ./five

In terminal-3 make connection to the server
T-3 $ tcp-echo-client 127.0.0.1 50000

Do not enter a string

In terminal-4 test the concurrent server
T-4 $ tcp-echo-client 127.0.0.1 50000

Did the server run correctly (Y / n)

Enter a string in terminal-3

Did the server run correctly ? (Y / n)

Check for zombie
T-3 $ ps ax

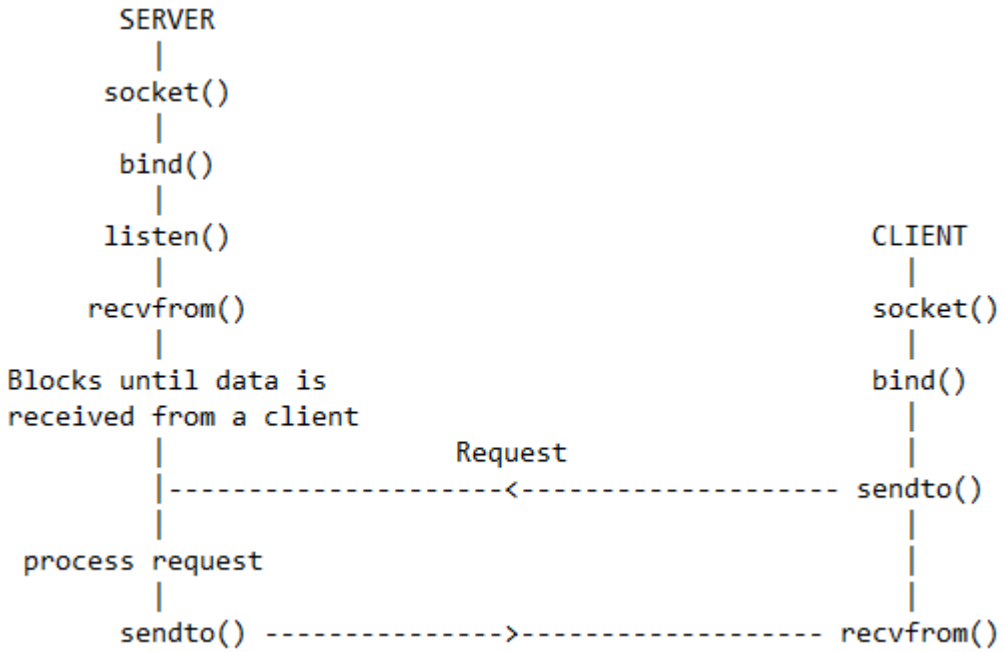
Was there zombies ? (Y / n)

Terminate server with CTRL-C
T-2 CTRL-C

T-3 $

End of Assignment-5.

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



[f6.c] is a UDP ECHO server. The calls made by the server are shown in the above figure.

// file-name f6.c
// UDP ECHO server using all interfaces

#include <sys/types.h>

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

#define  SERVER_PORT     50000
#define  BUFFERSIZE      512    // no more than this

extern int errno;

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

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

  // members of [server_addr] structure are initialised with the
  // address of the server
  server_addr.sin_family      =  AF_INET;
  server_addr.sin_addr.s_addr =  htonl(INADDR_ANY);
  server_addr.sin_port        =  htons(SERVER_PORT);

  // bind socket with IPv4 address of server
  n = bind( ser_sd, (struct sockaddr *) & server_addr, sizeof(server_addr) );

  clientlength = (socklen_t)sizeof(client_addr);
  while( 1 ) // endless loop
  {
    printf("%s: using UDP port-%u \n", argv[0], SERVER_PORT );
    memset( buffer, '\0', BUFFERSIZE );
    i = recvfrom( ser_sd, buffer, BUFFERSIZE,0, (struct sockaddr *) &client_addr,
&clientlength );
    if( i == -1 ) perror("recvfrom-call");

    // printing client's address and port are not essential
    printf("client's IP address = %s \n", inet_ntoa(client_addr.sin_addr)  );
    printf("client's port = %u \n", ntohs(client_addr.sin_port) );

    j = write( STDOUT_FILENO, "received from client-> ", 23 );
    if( j == -1 ) perror("write-stdout");

    printf("%s \n", buffer );

    j = sendto( ser_sd, buffer, i, 0, (struct sockaddr *)  &client_addr, clientlength);
    if( j == -1 ) perror("sendto-call");
  }  // end of while block
```

```
  close( ser_sd );
  exit(0);
}
```

Assignment-6:

Compile the program and start the server
T-2 $ gcc -Wall ./f6.c -o ./six
T-2 $ ./six

Test the server with [udp-echo-client] written by you in a previous experiment.
T-3 $ udp-echo-client 127.0.0.1   50000
T-3 $ udp-echo-client 192.168.5.___   50000

Did the server run correctly ? (Y / n)

End of Assignment-6.