BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI (RAJASTHAN)
IS F462 – Network Programming
Lab#6

Shared Memory:

1. Creating and attaching a shared memory segment

Shared memory is used to share the user memory among multiple processes with the help of kernel. The following program shows how to create and attach shared memory segment. The program is in shmget.c

```
main ()
{
  int id;
  key_t key;
  int shmid;
  char *data;

  key = ftok ("shmget.c", 'R');

  if ((shmid = shmget (key, 1024, 0644 | IPC_CREAT)) == -1)
    {
      perror ("shmget: shmget failed");
      exit (1);
    }
  data = shmat (shmid, (void *) 0, 0);

  if (data == (char *) (-1))
    perror ("shmat");

  return;
}
```

Q?

1. Compile and run the above program. The list of shared memory segments that can be viewed by your account can be listed by running the command `$ipcs –m`.
2. Run the command $ ipcs -m -i <shmid> and check what is the size of shared memory.
3. Write a line that adds some data to the shared memory. Something like, strcpy(data,"hello world");
4. Copy this program and modify to read and print the contents of shared memory. Verify whether you are getting the same contents are not.
5. Modify the above program as follows. The complete program is available in shmgetC.c

```
if((pid==fork())==0)
{

for(i=0;i<50;i++)
```

```
{      printf("%s", data);
sprintf(data, "I am child and my pid is %d and my count is %d\n", getpid(), i);
}
exit(0);
}
else if(pid>0)
{

for(i=0;i<50;i++)
{      printf("%s", data);
sprintf(data, "I am parent and my pid is %d and my count is %d\n", getpid(),
i);
}
}
```

Run the above program and see the output. Did it match you expectations. Justify? Find the solution.

6. Write a line that detaches the shared memory segment
7. Write a line that removes the shared memory segment
8. Design a program that verifies that removing a shared memory segment doesn't erase it from the memory unless all the processes have detached it.

2. Client and server using shared memory segments controlled by semaphores:

The following programs illustrate the use of shared memory for clients and servers running on the same system. This is a very popular use of shared memory. This example is taken from

http://publib.boulder.ibm.com/infocenter/iseries/v5r3/index.jsp?topic=%2Fapis%2Fapiexusmem.htm

Server program (shmserver.c)

Client program (shmclient.c)

Q?

1. Observe how the shared memory is protected using the semaphores
2. Make changes in the above code as per the following question: The program shmclient.c will writes an array of N numbers where N is occupied in first 1 byte and thereafter every two bytes stores each number. The server sums up all the numbers and writes the output next to the last number in 4 bytes. Client read the result written by the server.

Memory Mapping:

We have seen that fifos are basically named pipes. Similarly, mapped memory is like shared memory with a name. With this little bit of information, we can

conclude that there is some sort of relationship between files in the file system and memory mapping techniques - and, in fact, there is. Simply put, the memory mapping mechanism deals with the mapping of a file in the file system to a portion of memory through mmap() function.

```
#include <sys/mman.h>
Void *mmap(void *addr, size_t len, int prot, int flag, int filedes,
off_t off);
```

Consider the following program in which two processes will communicate through memory mapping. "Write.c" will write the data into some file and "read.c" will access the data using mmap() function.


write.c


read.c

TCP Socket Programming

netstat:

netstat is used for several purposes.

netstat –ni

displays all the interfaces on the system

netstat –r

displays routing table on the system

netstat –a

displays the status of all sockets on the system. t and u can be used to restrict socket display to tcp or udp sockets respectively

Look at the following screen shot. We will find the characteristic of all TCP connections in the system. The list shows that protocol name, size of data in send buffer, size of data in receive buffer, local end point address, remote end point address, and the state of the connection.

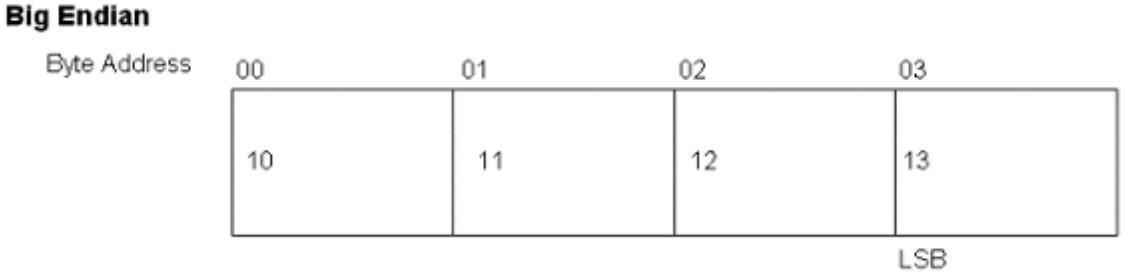
```
khari@prithvi:~$ netstat -nt
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 172.24.2.4:32929        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:46455        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47652        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47653        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47654        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47655        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47648        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47649        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47650        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47651        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47656        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47657        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47658        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47659        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47644        172.21.1.11:8080        TIME_WAIT
tcp        0      0 172.24.2.4:47645        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47646        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47647        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:47641        172.21.1.11:8080        TIME_WAIT
tcp        0      0 172.24.2.4:42691        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:54006        172.21.1.11:8080        ESTABLISHED
tcp        0      0 172.24.2.4:34034        172.30.3.3:22           ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.16.13.1:1145 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.16.18.:22110 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.16.10.:51481 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.16.10.:51472 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.24.4.50:2388 ESTABLISHED
tcp6       0    660 ::ffff:172.24.2.4:22    ::ffff:172.16.9.8:49205 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.24.4.50:2389 ESTABLISHED
tcp6       0      0 ::ffff:172.24.2.4:22    ::ffff:172.18.3.75:2221 ESTABLISHED
khari@prithvi:~$
```

Q?

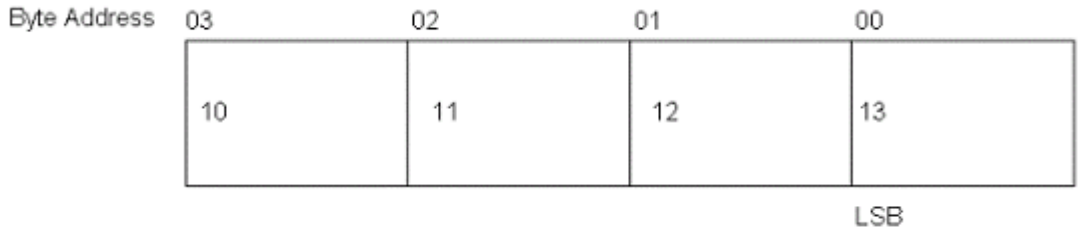
1. By looking at the out put of netstat, can you find out how many TCP connections are present in the system? Is it possible that number of connections can be less than the number of entries displayed?

Determining host byte order:

Consider a 16-bit integer that is made up of 2 bytes. There are two ways to store the two bytes in memory: with the low-order byte at the starting address, known as little-endian byte order, or with the high-order byte at the starting address, known as big-endian byte order. See the following figure where a 4 byte integer is stored in two different ways.



Little Endian



While transmitting piece of number data such as integers, it is important to have standard byte ordering. The sockets api follow big-endian style of storing. The following illustrates how one can find the way in which the local system is storing the numbers.

/*byteorder.c*/


byteorder.c

Q?

1. Find out the byte order used in your system?
2. Write a function `int ltob(int n)` which takes little-endian integer and converts to big-endian style?
3. Write a function `int btol(int n)` which takes big-endian integer and converts to little-endian style?
4. Explore the following set of functions. They are used to convert host byte order to network byte order and vice versa without actually knowing what the host byte order is. They are very much useful in network programming for they help in portability of the programs.
    a. ntohs() b. htons() //for 2 byte integers
    c. ntohl() d. htonl() //for 4 byte integers

inet_aton, inet_addr, inet_ntoa Functions:-
These functions convert an IPV4 address from a dotted-decimal string (eg. 172.24.2.4) to its 32 bit network byte ordered binary value.

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

1. int inet_aton (const char *strptr, struct in_addr *addrptr);
2. in_addr_t inet_addr(const char *strptr);
3. char *inet_ntoa(struct in_addr inaddr);

inet_pton and inet_ntop Functions:-
These functions works with both IPV4 and IPV6 addresses. ‘p’ and ‘n’ stands for presentation and numeric.

1. int inet_pton(int family, const char *strptr, void *addrptr);
2. const char* inet_ntop(int family, const void *addrptr, char *strptr, size_t len);


tcpclient.c


tcpserver.c

Q?

1. Run the server and type ”netstat –nt “ command in another terminal to see the state of server on the port(In this example: 12345) it is waiting for clients.
2. For testing above functions, remove comments from the corresponding line in tcpclient.c and tcpserver.c then compile and see the output. Execute client and server in different terminals.

Writing a simple TCP Client:
Consider the following TCP client. It gets the current date and time from a server running at port 13 and given ip address.

```
/*daytimeclient.c*/

#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define MAXLINE 80

typedef struct sockaddr SA;

int main(int argc, char **argv)
{
       int    sockfd, n;
       char   recvline[MAXLINE + 1];
       struct sockaddr_in   servaddr;
       if (argc != 2)
              {
              printf("usage: a.out <IPaddress>\n");
              exit(1);
              }
       if ( (sockfd = socket(PF_INET, SOCK_STREAM, 0)) < 0)  //creating socket
              perror("socket error");

       bzero(&servaddr, sizeof(servaddr));      //initializing addr structure to zero

       servaddr.sin_family = AF_INET;    //initializing add structure
       servaddr.sin_port   = htons(13);  /* daytime server runs at port 13*/

       if (inet_pton(AF_INET, argv[1], &servaddr.sin_addr) <= 0)
              printf("inet_pton error\n");


       if (connect(sockfd, (SA *) &servaddr, sizeof(servaddr)) < 0) //connecting
              perror("connect error");

       while ( (n = read(sockfd, recvline, MAXLINE)) > 0) {   //reading data from
server
```

```
                recvline[n] = 0;     /* null terminate */
                if (fputs(recvline, stdout) == EOF)            //putting to display
                        perror("fputs error");
        }
        if (n < 0)
                perror("read error");
        exit(0);
}
```

Q?

Errors involved in TCP connections:
Observe the steps involved in making a connection to a remote server.

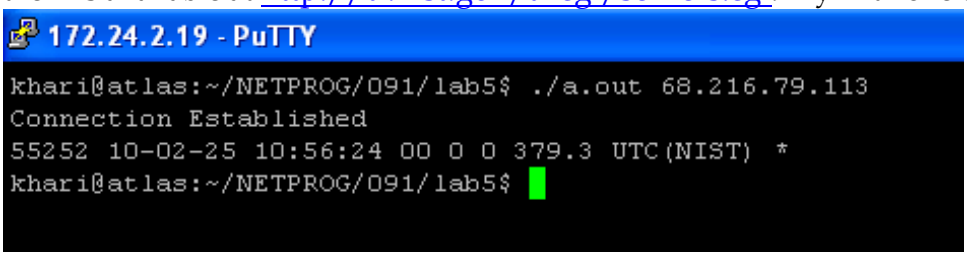
run the program with the following inputs

`./a.out 68.216.79.113`
It will take at least 4 mins to complete the above command because the host doesn't exist or is not reachable from within prithvi server. So TCP tries many times to retransmit the packets. Find out what error you would get? These kinds of errors are called as **SOFT errors** where TCP doesn't give up trying.

`./a.out 172.24.2.36`
Find out what error it would give? What does that error mean? These kinds of errors are called **HARD errors**. Here TCP immediately gives up trying.

`./a.out 172.24.2.1`
Find out what error it would give? What does that error mean?

There are no servers which are running daytime service at port 13 in our campus. But there are many time servers in internet running at port 13 to provide time synchronization. The list of them is available at http://tf.nist.gov/tf-cgi/servers.cgi. Try with one of the IPs given there.



In the above program change the port number to 25 and give ip 172.24.2.80 as input. What output did you get? What is your understanding of this?

In terms of the TCP state transition diagram, connect moves from the CLOSED state (the state in which a socket begins when it is created by the socket function) to the SYN_SENT state, and then, on success, to the ESTABLISHED state. If connect fails, the socket is no longer usable and must be closed.

Simple client-server program:

We will explain the aspects to be noted while writing client and server using echo client and server example. This material is taken from the book 'TCP IP Sockets in C: practical guide". Client sends a string to the server and server will return the same string back to the client.

There are 4 files given below.

/*DieWithError.c*/

```
#include <stdio.h>
#include <stdlib.h>

void DieWithError(char *errorMessage)
{
    perror(errorMessage);
    exit(1);
}
```

/*Client.c*/

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

#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in echoServAddr;
    unsigned short echoServPort;
    char *servIP;
    char *echoString;
    char echoBuffer[RCVBUFSIZE];
    unsigned int echoStringLen;
    int bytesRcvd, totalBytesRcvd;

    if ((argc < 3) || (argc > 4))        {
       fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n",
               argv[0]);
       exit(1);
    }

    servIP = argv[1];
    echoString = argv[2];

    if (argc == 4)
        echoServPort = atoi(argv[3]);
```

```
    else
        echoServPort = 7;


    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");


    memset(&echoServAddr, 0, sizeof(echoServAddr));

    echoServAddr.sin_family      = AF_INET;
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);
    echoServAddr.sin_port        = htons(echoServPort);


    if (connect(sock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("connect() failed");

    echoStringLen = strlen(echoString);


    if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
        DieWithError("send() sent a different number of bytes than
expected");

    totalBytesRcvd = 0;

    printf("Received: ");
    while (totalBytesRcvd < echoStringLen)
    {

        if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1, 0)) <= 0)
            DieWithError("recv() failed or connection closed prematurely");
        totalBytesRcvd += bytesRcvd;
        echoBuffer[bytesRcvd] = '\0';
        printf("%s", echoBuffer);
    }

    printf("\n");

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

/*EchoServer.c*/

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

#define MAXPENDING 5
```

```
void DieWithError(char *errorMessage);
void HandleTCPClient(int clntSocket);

int main(int argc, char *argv[])
{
    int servSock;
    int clntSock;
    struct sockaddr_in echoServAddr;
    struct sockaddr_in echoClntAddr;
    unsigned short echoServPort;
    unsigned int clntLen;

    if (argc != 2)
    {
        fprintf(stderr, "Usage:  %s <Server Port>\n", argv[0]);
        exit(1);
    }

    echoServPort = atoi(argv[1]);


    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");


    memset(&echoServAddr, 0, sizeof(echoServAddr));

    echoServAddr.sin_family = AF_INET;

    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    echoServAddr.sin_port = htons(echoServPort);


    if (bind(servSock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("bind() failed");


    if (listen(servSock, MAXPENDING) < 0)
        DieWithError("listen() failed");

    for (;;)
    {

        clntLen = sizeof(echoClntAddr);


        if ((clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr,
                               &clntLen)) < 0)
            DieWithError("accept() failed");



        printf("Handling client %s\n", inet_ntoa(echoClntAddr.sin_addr));

        HandleTCPClient(clntSock);
    }
}
```

/*HandleTcpClient.c*/

```
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>

#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);

void HandleTCPClient(int clntSocket)
{
    char echoBuffer[RCVBUFSIZE];
    int recvMsgSize;


    if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
        DieWithError("recv() failed");


    while (recvMsgSize > 0)
    {

        if (send(clntSocket, echoBuffer, recvMsgSize, 0) != recvMsgSize)
            DieWithError("send() failed");


        if ((recvMsgSize = recv(clntSocket, echoBuffer, RCVBUFSIZE, 0)) < 0)
            DieWithError("recv() failed");
    }

    close(clntSocket);
}
```

Q?

1. compile the above code as follows
    a. gcc –o echocli EchoClient.c DieWithError.c
    b. gcc –o echoser EchoServer.c HandleTCPClient.c DieWIthError.c
2. run the server and client in separate windows with appropriate arguments. Let us understand the program by reading the comments that are elaborately given against each block of code. One thing that is noticeable here is that send() and recv() are used for sending and receiving data. When the client is receiving data the recv() is used in a loop. This is because TCP is byte-stream protocol. One implication of this type of protocol is that send() boundaries are not preserved. The bytes sent by a call to send() on one end of connection may not all be returned by a single call to recv() on the other end. So we need to repeatedly receive bytes until we have received as many as we sent. In all likelihood, this loop will only be executed once because the data from the server will in fact be returned all at once; however, that is not guaranteed to happen, and so we have to allow for the possibility that multiple reads are required. **This is a basic principle of writing applications that use sockets: you must never assume anything about what the network and the program at the other end are going to do**.
3. Now type netstat in the client terminal. You will notice that now your server is also listed among the list of open connections. Try to identify the server using the port that is

assigned to it. Also make a note of the state in which the server currently is and the local and foreign addresses assigned to it.

4. Edit the file client.c by adding a scanf() statement after the connect statement. Now run server and client on seperate terminals. Let the client wait for an input at the scanf() statement. In a third terminal type netstat an observe the states of server and client. Also make a note of the addresses assigned to the server and client.
5. Run the above modified client, and while it is waiting at scanf(), terminate the client by pressing Ctrl-Z and type netstat immidiately. You should notice that the client is in TIME-WAIT state.
6. Right now the client takes only one string at a time from user and sends it to the server. Modify the client to take as many strings as entered by user terminated Ctrl-D.
7. In extension to modified client in 6, send the strings in **batch mode.** And observe whether you receive the replies corresponding to the requests. What data you need to preserve in order to make the batch-mode client work error free.
8. can you think of a way to handle sending requests and receiving replies simultaneously in batch-mode client?
9. Coming to Server, can you identify whether the server is iterative or concurrent?
10. Modify the above given server to a concurrent server i.e. handling multiple clients simultaneously by forking a child for each client connection
11. Let us simulate a boundary condition.
    a. Client has already communicated a set of strings to server and it is waiting for further input from user
    b. Kill your server by kill -9 <server pid>
    c. See what would happen by using netstat. Observe the states of server socket and client socket.
12. In the above example if we modify the client to be in batch mode and simultaneously sending and receiving data, then it becomes difficult on server side to determine what is sent by the client as "one string". To make this happen, the string sent by client needs to be either having fixed width or having to end with a special character(s). The server will check for these special characters and reply only after reading the whole string. This will also help the client to distinguish between two replies of a server corresponding to two requests of the client.

Errors involved due to abnormal Termination of server:

```
/*sigPipeClient.c*/
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <signal.h>
#define RCVBUFSIZE 32

void DieWithError(char *errorMessage);


void handler(int signo)
{
printf("SIG PIPE received");
}
```

```
int main(int argc, char *argv[])
{
    void handler(int);
    signal(SIGPIPE,handler);
    int sock;
    struct sockaddr_in echoServAddr;
    unsigned short echoServPort;
    char *servIP;
    char *echoString;
    char echoBuffer[RCVBUFSIZE];
    unsigned int echoStringLen;
    int bytesRcvd, totalBytesRcvd;


    if ((argc < 3) || (argc > 4))        {
       fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo Port>]\n",
               argv[0]);
       exit(1);
    }

   servIP = argv[1];
    echoString = argv[2];

    if (argc == 4)
        echoServPort = atoi(argv[3]);
    else
        echoServPort = 7;

    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");
    memset(&echoServAddr, 0, sizeof(echoServAddr));
    echoServAddr.sin_family      = AF_INET;
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);
    echoServAddr.sin_port        = htons(echoServPort);


    if (connect(sock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("connect() failed");
    char sendLine[100],recvLine[100];
    printf("pid: %d",getpid());
    while(fgets(sendLine,sizeof(sendLine),stdin)!=NULL)
    {
    send(sock,sendLine,strlen(sendLine),0);
    sleep(1);
    send(sock,sendLine,strlen(sendLine),0);
        if ((bytesRcvd = recv(sock, recvLine, 99, 0)) <= 0)
        {
        perror("recv error");
        }
    puts(recvLine);
    }
      return(0);
}
```

Q?

1. start the TCP server and the above client, and type one word to the client to verify that all is ok.
2. Find the process id of the server and kill it. This causes a FIN to be sent to the client and the TCP responds with an ACK.
3. Nothing happens to the client, the client TCP receives the FIN from the server TCP and responds with an ACK but the problem is the client process blocked in the fgets();
4. run netstat at this time. Server will be in FIN_WAIT2 state and client will be in CLOSE_WAIT state.
5. Try sending one more word to the server from the client. A SIGPIPE signal should be received.

===End of Lab6===