Lab 3 : Socket Programming Tutorial

Esha Desai

USC ID: 6993245898

CONTENTS

1. Client-server model with enhancement……..3
2. Single Process Concurrent Server Using Select() syscall……8

Sockets and File Descriptors:

Sockets are API (Application Programming Interface) for TCP/IP Protocol stack and they are used for bidirectional inter-process communication. All the programs on Unix work as files and so there is an integer associated with every file to help in communication within the programs.

1.Client-server model with enhancement:

I tried running the basic example code provided in the tutorial using the simple read() and write() syscalls. Then I made the changes in the code for receiving and sending data using the send() and recv() syscalls according to the TCP protocol. As asked in the tutorial, the enhancements using the dostuff() function with the new socket file descriptor as an argument, were made.
The NS script is pretty straightforward . There are two nodes :A for server and B for client.

NS Script:

```
# This is a simple ns script. Comments start with #.
set ns [new Simulator]
source tb_compat.tcl
set nodeA [$ns node]
set nodeB [$ns node]
set link0 [$ns duplex-link $nodeB $nodeA 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
# Set the OS on a couple.
tb-set-node-os $nodeA FBSD-STD
tb-set-node-os $nodeB RHL-STD
$ns rtproto Static
# Go!
$ns run
```

Following is the enhanced code working on TCP protocol with the dostuff function outside the main function.

SERVER:

```
/* *A simple server in the internet domain using TCP
   The port number is passed as an argument */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
```

```
#include <netinet/in.h>

void error(const char *msg)
{
   perror(msg);
   exit(1);
}
void dostuff(int newsockfd)
{
            //socklen_t clilen;
            //struct sockaddr_storage cli_addr;
            char buffer[256];
            int n;
             bzero(buffer,256);
             n = recv(newsockfd,buffer,255,0);
             if (n < 0) error("ERROR reading from socket");
             printf("Here is the message: %s\n",buffer);
             n = send(newsockfd,"I got your message",18,0);
             if (n < 0) error("ERROR writing to socket");
}


int main(int argc, char *argv[])
{
            socklen_t clilen;
            struct sockaddr_storage cli_addr;
    int sockfd, newsockfd, portno;
    struct sockaddr_in serv_addr;
            pid_t pid;
    char buffer[256];

    int n;
    if (argc < 2) {
       fprintf(stderr,"ERROR, no port provided\n");
       exit(1);
    }
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
      error("ERROR opening socket");
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = atoi(argv[1]);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    serv_addr.sin_port = htons(portno);
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
          sizeof(serv_addr)) < 0)
```

```
        error("ERROR on binding");
    listen(sockfd,5);
    clilen = sizeof(cli_addr);
            while (1)
            {
             newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
             if (newsockfd < 0)
                    error("ERROR on accept");
             pid = fork();
             if (pid < 0)
                    error("ERROR on fork");
             if (pid == 0) //new child process
             {
                    close(sockfd);
                    dostuff(newsockfd);
                    exit(0);
             }
             else //parent process
                    close(newsockfd);
            } /* end of while */

            return 0;
}
```

CLIENT:

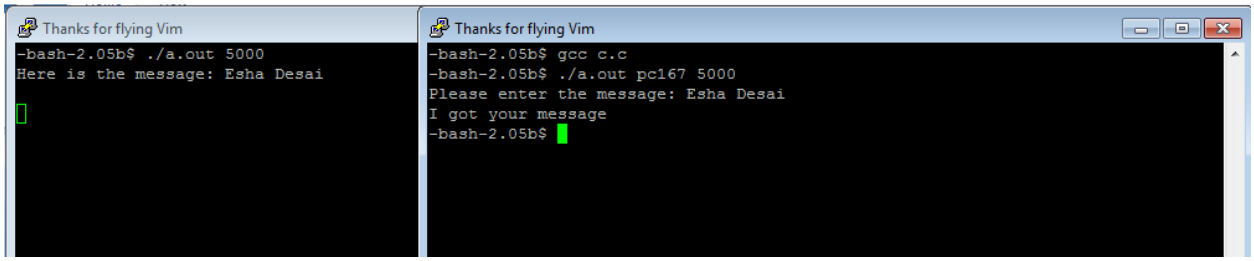
```
//*TCP CLIENT
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
   perror(msg);
   exit(0);
}

int main(int argc, char *argv[])
{
   int sockfd, portno, n;
   struct sockaddr_in serv_addr;
   struct hostent *server;
```

```
        socklen_t serv_len;
    char buffer[256];
    if (argc < 3) {
      fprintf(stderr,"usage %s hostname port\n", argv[0]);
      exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
       error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
       fprintf(stderr,"ERROR, no such host\n");
       exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    serv_len=sizeof (serv_addr);
    if (connect(sockfd,(struct sockaddr *) &serv_addr,serv_len) < 0)
       error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = send(sockfd,buffer,strlen(buffer),0);
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n = recv(sockfd,buffer,255,0);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    close(sockfd);
    return 0;
}
```

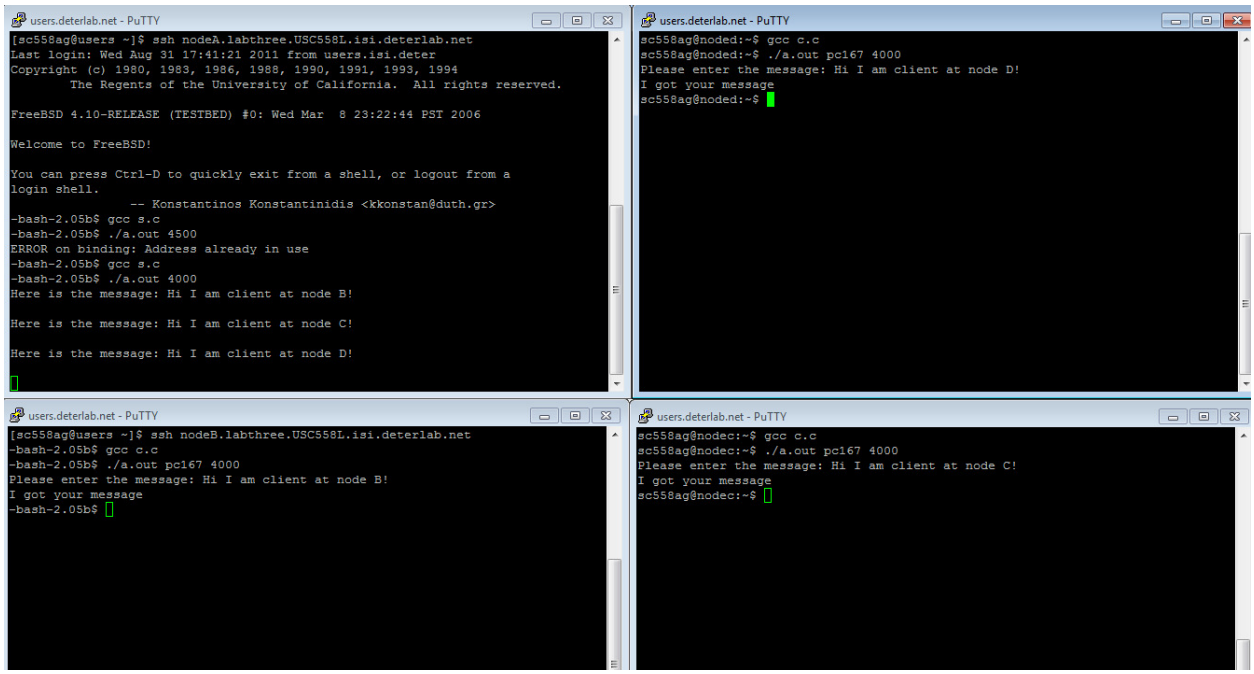
Working on just 2 nodes , one for server and one for client.



NS Script: (for creating 4 nodes)

```
# This is a simple ns script. Comments start with #.
set ns [new Simulator]
source tb_compat.tcl
set nodeA [$ns node]
set nodeB [$ns node]
set nodeC [$ns node]
set nodeD [$ns node]
set link0 [$ns duplex-link $nodeB $nodeA 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
set link1 [$ns duplex-link $nodeC $nodeA 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
set link2 [$ns duplex-link $nodeD $nodeA 30Mb 50ms DropTail]
tb-set-link-loss $link0 0.01
# Set the OS on a couple.
tb-set-node-os $nodeA FBSD-STD
tb-set-node-os $nodeB RHL-STD
$ns rtproto Static
# Go!
$ns run
```

I put the server code on node A and client code on Node B, C and D. On running the code from respective nodes, the following output was obtained. From the details of the experiment I found that the physical node mapping showed that the physical address of node A was pc 167.



2.Single Process Concurrent Server Using Select() syscall:

A single server can handle multiple clients at a time using select syscall. The select() system call is helpful to observe multiple sockets simultaneously instead of just one. The main advantage of select syscall is that while the server remains blocked by just one client , it can still monitor other sockets for other clients trying t connect. The server listens on one socket and if it gets a new connection it makes a new socket file descriptor for the client to carry out further exchange of data. While it exchanges data on the new socket , it still monitors other socket file descriptors for a new connection on the listening socket and for exchange of data on the other sockets. Below is the code for server and the client.

Server code:

```
#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>
#include <netdb.h>
#include <errno.h>
#include <sys/wait.h>
#include <signal.h>
//#define PORT "5501"   // port we're listening on
#define MAXBUFLEN 255

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)
{
   if (sa->sa_family == AF_INET) {
      return &(((struct sockaddr_in*)sa)->sin_addr);
   }
   return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
void sigchld_handler(int s)//for reaping all zombie processes
{
   while(waitpid(-1, NULL, WNOHANG) > 0);
}

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

{
   fd_set master;    // master file descriptor list
   fd_set read_fds;  // temp file descriptor list for select()
    int fdmax;        // maximum file descriptor number
    int listener;     // listening socket descriptor
    int newfd;        // newly accept()ed socket descriptor
    struct sockaddr_storage remoteaddr; // client address
    socklen_t addrlen;
    socklen_t addr_len;
  // struct sockaddr_storage their_addr;
    char bufr[MAXBUFLEN];    // buffer for receiving
    char bufs[MAXBUFLEN];                                 //buffer for sending
    int nbytes,numbytes,recbytes;
    char remoteIP[INET6_ADDRSTRLEN];
    char s[INET6_ADDRSTRLEN];
    int yes=1;        // for setsockopt() SO_REUSEADDR, below
    int i,l, j, rv,count;
    struct addrinfo hints, *ai, *p;

    FD_ZERO(&master);    // clear the master and temp sets
    FD_ZERO(&read_fds);

    // get us a socket and bind it
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_UNSPEC;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;
    if ((rv = getaddrinfo(argv[1], argv[2], &hints, &ai)) != 0) {
```

```
        fprintf(stderr, "selectserver: %s\n", gai_strerror(rv));
        exit(1);
    }

    for(p = ai; p != NULL; p = p->ai_next) {

        listener = socket(p->ai_family, p->ai_socktype, p->ai_protocol);
         if (listener < 0) {
            continue;
         }
      // lose the "address already in use" error message
        setsockopt(listener, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int));
        if (bind(listener, p->ai_addr, p->ai_addrlen) < 0) {
            close(listener);
            continue;
        }
        break;

    }
    // if we got here, it means we didn't get bound
    if (p == NULL) {
        fprintf(stderr, "selectserver: failed to bind\n");
        exit(2);
    }
    freeaddrinfo(ai); // all done with this

    // listen
    if (listen(listener, 10) == -1) {
        perror("listen");
        exit(3);
    }
sa.sa_handler = sigchld_handler; // reap all dead processes
    sigemptyset(&sa.sa_mask);
    sa.sa_flags = SA_RESTART;
    if (sigaction(SIGCHLD, &sa, NULL) == -1) {
        perror("sigaction");
        exit(1);
    }

    printf("\nServer waiting for client.....\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

    // main loop
    for(;;)
{
```

```
read_fds = master; // copy it
  if (select(fdmax+1, &read_fds, NULL, NULL, NULL) == -1) {
     perror("select");
     exit(4);
  }
   // run through the existing connections looking for data to read
  for(i = 0; i <= fdmax; i++)
 {
     if (FD_ISSET(i, &read_fds))
 { // we got one!!
         if (i == listener)
        {
            // handle new connections
            addrlen = sizeof remoteaddr;
            newfd = accept(listener,(struct sockaddr *)&remoteaddr,&addrlen);

            if (newfd == -1)
           {
               perror("accept");
            }
          else
          {

               FD_SET(newfd, &master); // add to master set
               if (newfd > fdmax) {    // keep track of the max
                  fdmax = newfd;
               }
               printf("\nServer: new connection from %s on socket %d\n",

            inet_ntop(remoteaddr.ss_family,get_in_addr((struct sockaddr*)&remoteaddr), remoteIP,
          INET6_ADDRSTRLEN),newfd);

            }
         }
      else
      {
            // handle data from a client
         printf("server: waiting to recv ...\n");
         //RECEIVE ACTUAL DATA
        if ((nbytes = recv(i, bufr, sizeof (bufr), 0)) <= 0)
         {
        // got error or connection closed by client
       if (nbytes == 0)
      {
     // connection closed
        printf("selectserver: socket %d hung up\n", i);
        }
      else
       {
        perror("recv");
       }
```

```
            close(i); // bye!
          FD_CLR(i, &master); // remove from master set
          }

else
{
printf("\nserver: receiving data from Client\n");
printf("Got packet from client\n");
printf("data is:");
for(i=0;i<MAXBUFLEN;i++)
{
printf("%c",bufr[i]);
}
printf("\n\n");
}
  } // END handle data from client
  } // END got new incoming connection
 } // END looping through file descriptors
} // END for(;;)--and you thought it would never end!
    return 0;

}
```

Client Code:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <strings.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#define MAXBUFSIZE 255 // max number of bytes we can SEND/get at once

// get sockaddr, IPv4 or IPv6:
void *get_in_addr(struct sockaddr *sa)

{
    if (sa->sa_family == AF_INET) {

        return &(((struct sockaddr_in*)sa)->sin_addr);
    }
    return &(((struct sockaddr_in6*)sa)->sin6_addr);
}
```

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

   int sockfd, numbytes;
   char bufr[MAXBUFSIZE];
   char bufs[MAXBUFSIZE];
   struct addrinfo hints, *servinfo, *p;
   int rv;
   char s[INET6_ADDRSTRLEN];

   if (argc != 3) {
      fprintf(stderr,"usage: client hostname\n");
      exit(1);
   }

memset(&hints, 0, sizeof hints);
hints.ai_family = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
if ((rv = getaddrinfo(argv[1], argv[2], &hints, &servinfo)) != 0) {
 fprintf(stderr, "getaddrinfo: %s\n", gai_strerror(rv));
 return 1;
}
// loop through all the results and connect to the first we can
for(p = servinfo; p != NULL; p = p->ai_next)
{
 if ((sockfd = socket(p->ai_family, p->ai_socktype,p->ai_protocol)) == -1)
 {
  perror("client: socket");
  continue;
 }
 if (connect(sockfd, p->ai_addr, p->ai_addrlen) == -1)
{
close(sockfd);
perror("client: connect");
continue;
 }
 break;
}
if (p == NULL)
{
fprintf(stderr, "client: failed to connect\n");
 return 2;
}
inet_ntop(p->ai_family, get_in_addr((struct sockaddr *)p->ai_addr),s, sizeof s);
printf("client: connecting to %s\n",s);
printf("Please enter the message: ");
   bzero(bufs,256);
   fgets(bufs,255,stdin);
if (send(sockfd, bufs, sizeof(bufs), 0) == -1)
{
```

```
perror("send");
}
if (recv(sockfd, bufr, sizeof(bufr), 0) == -1)
{
perror("send");
}
return 0;

}
```

Code for reaping all zombie processes:

Function before the main() function:

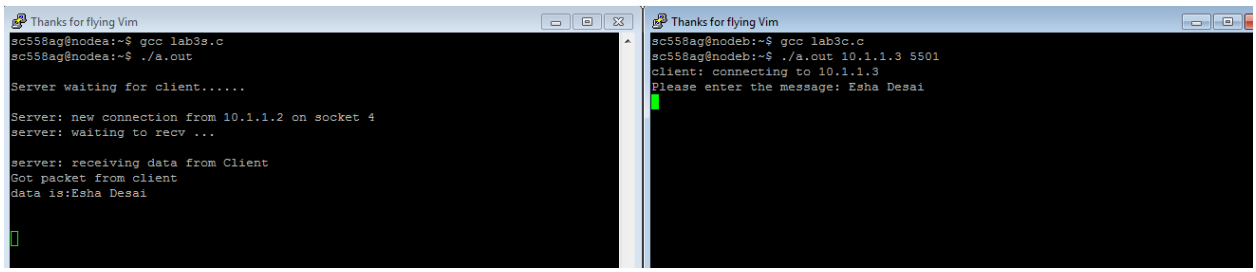
```
void sigchld_handler(int s)//for reaping all zombie processes
{
   while(waitpid(-1, NULL, WNOHANG) > 0);
}
```

We need to include <signal.h> and <sys/wait.h> header files for execution of the sigaction function.

Included in the main() function:

```
sa.sa_handler = sigchld_handler; // reap all dead processes
   sigemptyset(&sa.sa_mask);
   sa.sa_flags = SA_RESTART;
   if (sigaction(SIGCHLD, &sa, NULL) == -1) {
      perror("sigaction");
      exit(1);
   }
```

Running the code on just **two nodes**:



Using 4 nodes (3 clients trying to connect to 1 server):

The following are the details about the IP addresses of the 4 nodes A, B, C and D. Looking at this details we can tell the server which IP address it uses by the argument. And the clients also need to know the Ip address of the server for successful connection to the server. In this case the Ip address of the server

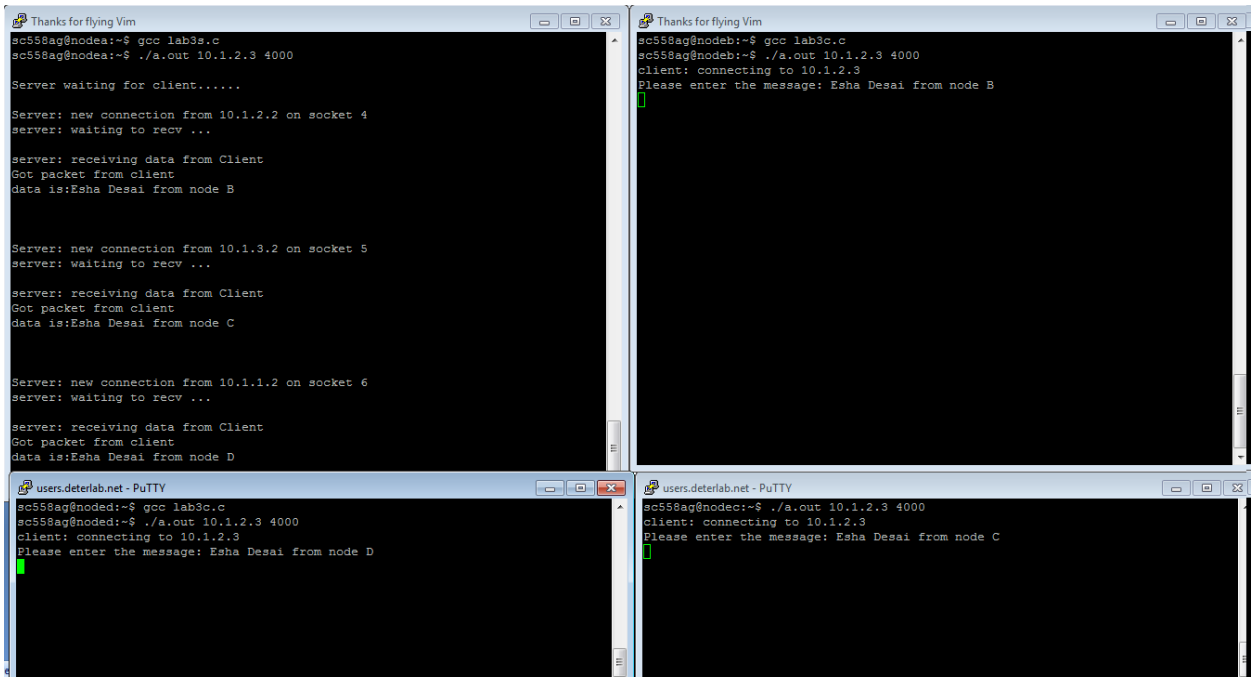
(here node A) is *10.1.2.3*. Also along with the IP address the server port address is given along with the arguments (instead of hard coding it in the code) while running the server and the client codes so that it is made sure that they are communicating on the same port.

```
Virtual Lan/Link Info:
ID              Member/Proto    IP/Mask         Delay     BW (Kbs)  Loss Rate
--------------- --------------- --------------- --------- --------- ---------
link0           nodeA:0         10.1.2.3        25.00     30000     0.00501256
                ethernet        255.255.255.0   25.00     30000     0.00501256
link0           nodeB:0         10.1.2.2        25.00     30000     0.00501256
                ethernet        255.255.255.0   25.00     30000     0.00501256
link1           nodeA:1         10.1.3.3        25.00     30000     0.00000000
                ethernet        255.255.255.0   25.00     30000     0.00000000
link1           nodeC:0         10.1.3.2        25.00     30000     0.00000000
                ethernet        255.255.255.0   25.00     30000     0.00000000
link2           nodeA:2         10.1.1.3        25.00     30000     0.00000000
                ethernet        255.255.255.0   25.00     30000     0.00000000
link2           nodeD:0         10.1.1.2        25.00     30000     0.00000000
                ethernet        255.255.255.0   25.00     30000     0.00000000

Physical Lan/Link Mapping:
ID              Member          IP              MAC                 NodeID
--------------- --------------- --------------- ------------------- ---------
link0           nodeA:0         10.1.2.3        00:04:23:c5:d5:32   pc145
                                                0/1 <-> 2/21        Nortel18ne
link0           nodeB:0         10.1.2.2        00:04:23:c5:dc:bc   pc192
                                                0/1 <-> 6/17        Nortel18ne
link1           nodeA:1         10.1.3.3        00:04:23:c5:d5:34   pc145
                                                2/1 <-> 2/24        Nortel18ne
link1           nodeC:0         10.1.3.2        00:04:23:c7:a6:8c   pc167
                                                0/1 <-> 4/13        Nortel18ne
link2           nodeA:2         10.1.1.3        00:04:23:c5:d5:33   pc145
                                                1/1 <-> 2/22        Nortel18ne
link2           nodeD:0         10.1.1.2        00:04:23:c5:de:60   pc168
                                                0/1 <-> 4/17        Nortel18ne
```

After running the client nodes from nodes B, C , D we can see that the server is able to make successful connection with all of the nodes. Here we can see that the client at node B is exchanging data on socket 4 and similarly client at node C on socket 5 and client at node D on socket 6.

Following is the snapshot of the output of the above server and client codes.



After making a 'make file':

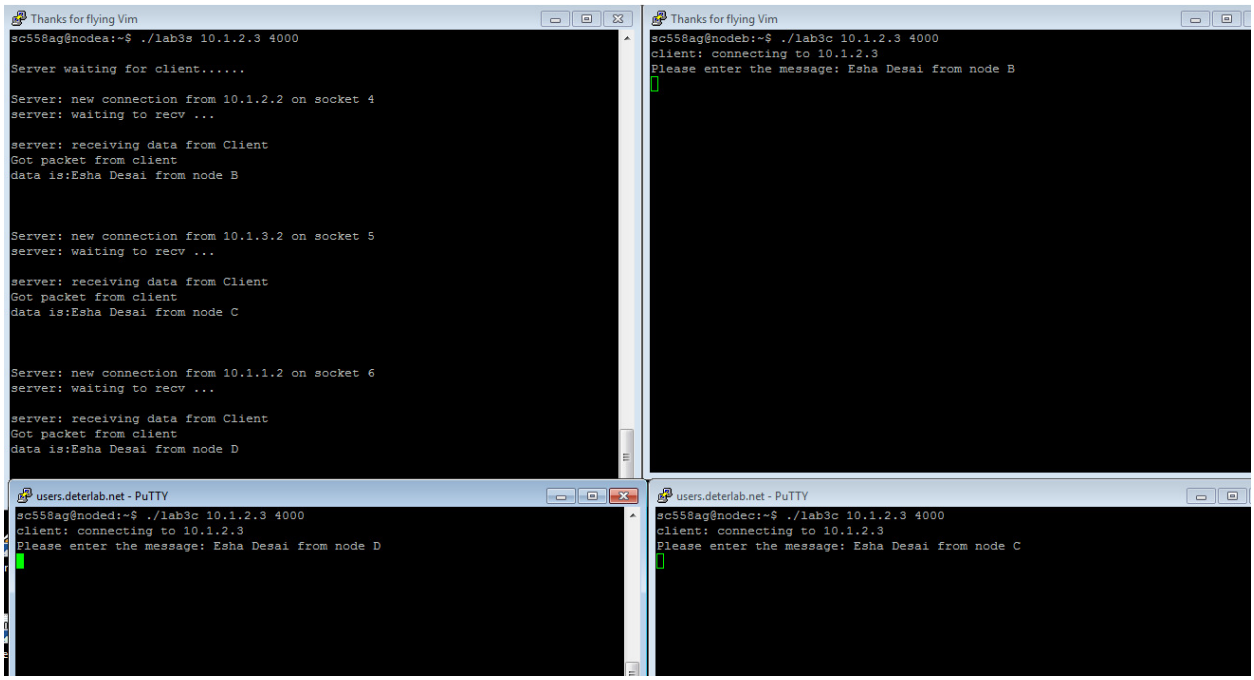
```
# This is a makefile
  all:    lab3s lab3c
  lab3s: lab3s.o
      gcc -o lab3s lab3s.o

  lab3s.o: lab3s.c
      gcc -c lab3s.c
  lab3c: lab3c.o
     gcc -o lab3c lab3c.o

 lab3c.o: lab3c.c
     gcc -c lab3c.c

 clean:
    rm -rf *.o lab3s lab3c
```

Make file makes compiling easier. Compiling of every code can be done simultaneously. And if the code of only one of the multiple files is changed then only the code which is modified is compiled.



Conclusion:

The tutorial was very helpful for me to get a hands-on experience before the projects start. I learnt the in – depth functioning of sockets and about the various system calls and their functions in socket programming. I learnt about the different types of servers, how multiple sockets can be handled by the server. I also learnt that different Operating systems behave differently to same codes. I tried running the code first on Ubuntu locally and then I tried to run them on Deter nodes. There were many distinctions in the code running on ubuntu and deter which I had to take care of which I learnt from this lab tutorial.