Operating Systems - EECE.5730

Instructor: *Prof. Dalila Megherbi*

Assignment – 4

Final Project-Part1

Due by 11-28-16

By,

- *Naga Ganesh Kurapati*

1) Objective:

The purpose of this project is to mimic distributed scheduler and remote procedure call using sockets. We need to design client- server application communicating using UDP sockets and client sends the commands to server, in response server executes the command and sends the output of the command.

2) Background:

Using sockets two process can communicate to exchange information over the network. Sockets are abstraction provided to the user to send or receive data over the network. They two well-known transport layer protocols namely UDP and TCP. TCP is reliable protocol whereas UDP is not. In this project we used the UDP socket. On the server side, it needs to have fixed IP addresses and Port number, so that client can initiate the connection using the information. So we need to bind the port number with given socket. Remote procedure call is nothing but initiating procedure (calling a function) on server side while sending the proper commands over the network. Server interpret the command, execute the commands and sends the output to client over the network. Whereas distributed scheduler works in similar way.

3) Algorithms/Functions used:

Project-Part1:

socket() – To open a socket
close() – To close the socket
bind() – To bind the socket with given port number
listen() – To on a socket
sendto() – To send the data to other application/machine using socket
recvfrom() - To receive the data to other application/machine using socket
inet_ntop() – To extract the IP address to given data buffer
fork() – To create a child process
Wait() – To wait for the child process to complete
strcpy() – To copy char data between array buffers
bzero() – To flush the given data buffer
strcmp() – To compare two strings, returns 0 if both strings are equal
system() – To execute the system commands from the C code
fopen() – To open a file on the system
fseek() – To set the position of read position in a file
ftell() – To return the current position in the file read
fread() – To read a file from disk
fclose() – To close the file
chdir() – To change current directory
setsockopt() – To socket time out
gethostbyname() – To get the IP address of the host

htons() – To convert given short to network byte format

User defined functions:

Report_error() – To print the error and exit the program
intializeSocket() – To initialize the socket on server side
GetRequest() – To receive data from socket
SendReply() – To send data to the socket
DoOperation() – To send the data to the server and receive the response

Chat Application:

pthread_create() – to create pthreads
pthread_join() – to wait for threads to complete and join
socket() – To open a socket
close() – To close the socket
bind() – To bind the socket with given port number
listen() – To on a socket
sendto() – To send the data to other application/machine using socket
recvfrom() - To receive the data to other application/machine using socket
setsockopt() – To socket time out
gethostbyname() – To get the IP address of the host
htons() – To convert given short to network byte format
strcpy() – To copy char data between array buffers
bzero() – To flush the given data buffer
strcmp() – To compare two strings, returns 0 if both strings are equal

User defined functions:

Report_error() – To print the error and exit the program
intializeSocket() – To initialize the socket on server side
GetRequest() – To receive data from socket
SendReplyServer() – To send data to the client from server
*SendReply() – To send data using pthreads
*GetRequest() – To get data using pthreads
SendReply() - To send data to the socket

4) Results:

Server is running on the anaconda 25. Three clients running on anaconda 10, 19,30.
First start the server, connection is initialized from the client by typing user name.
Here user1, user2, user3. Server prints the assigned port number to clients as shown in the fig. below.

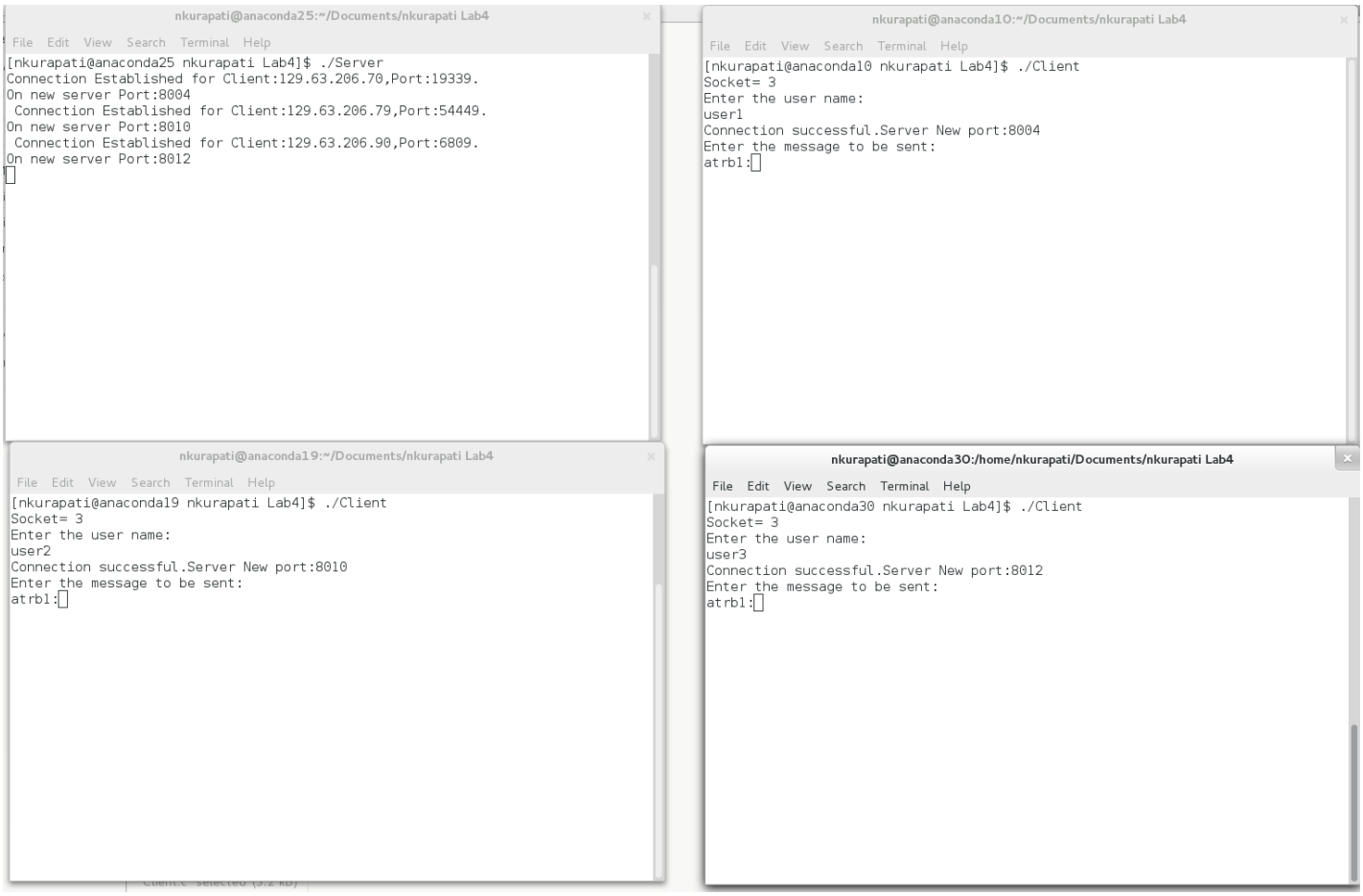

Fig.1 – Connection establishment

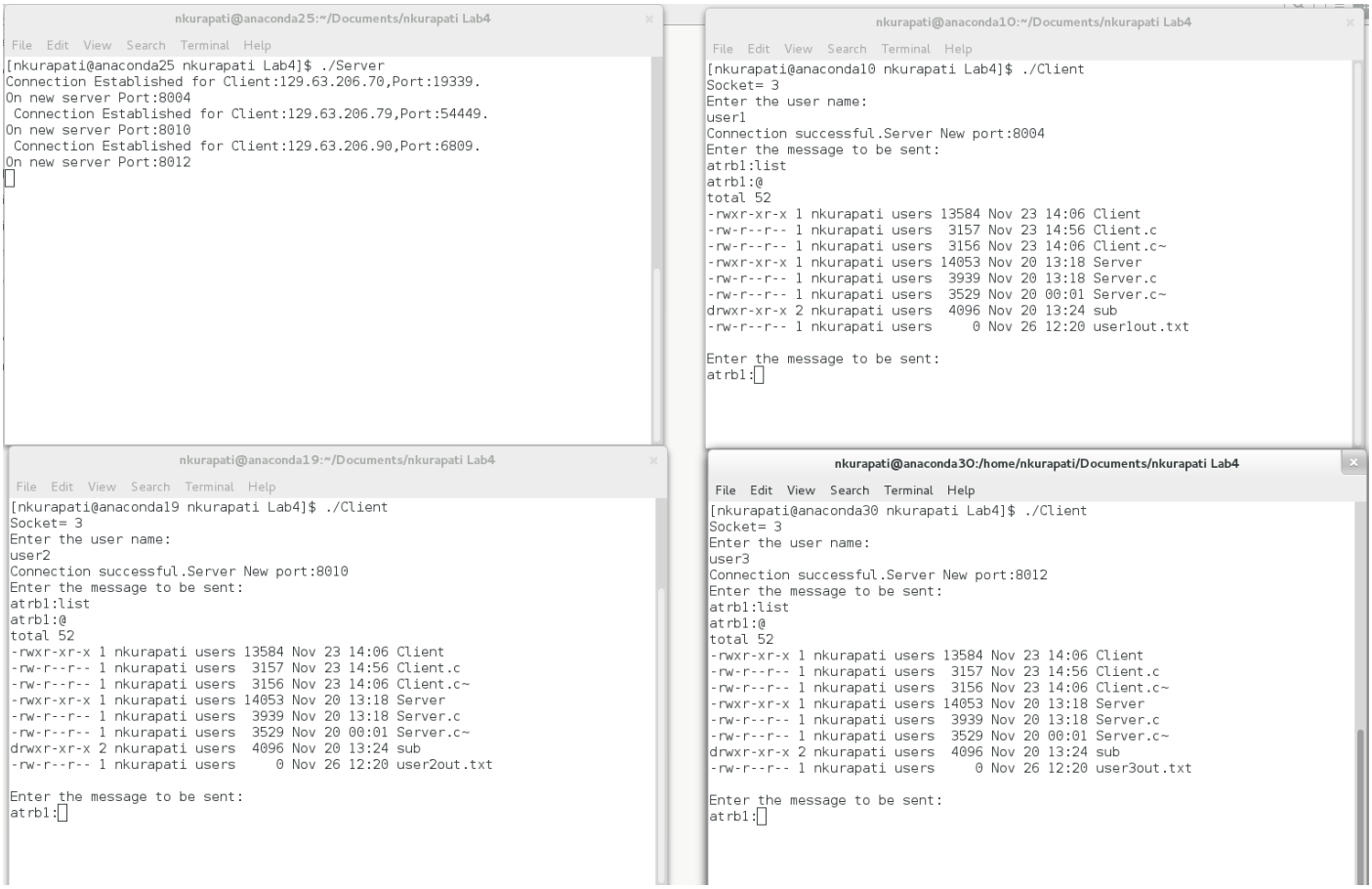

Fig.2 – Demonstrating the list command

You can see the first attribute typed is list and second can be anything. It list the contents of the directory on other side.

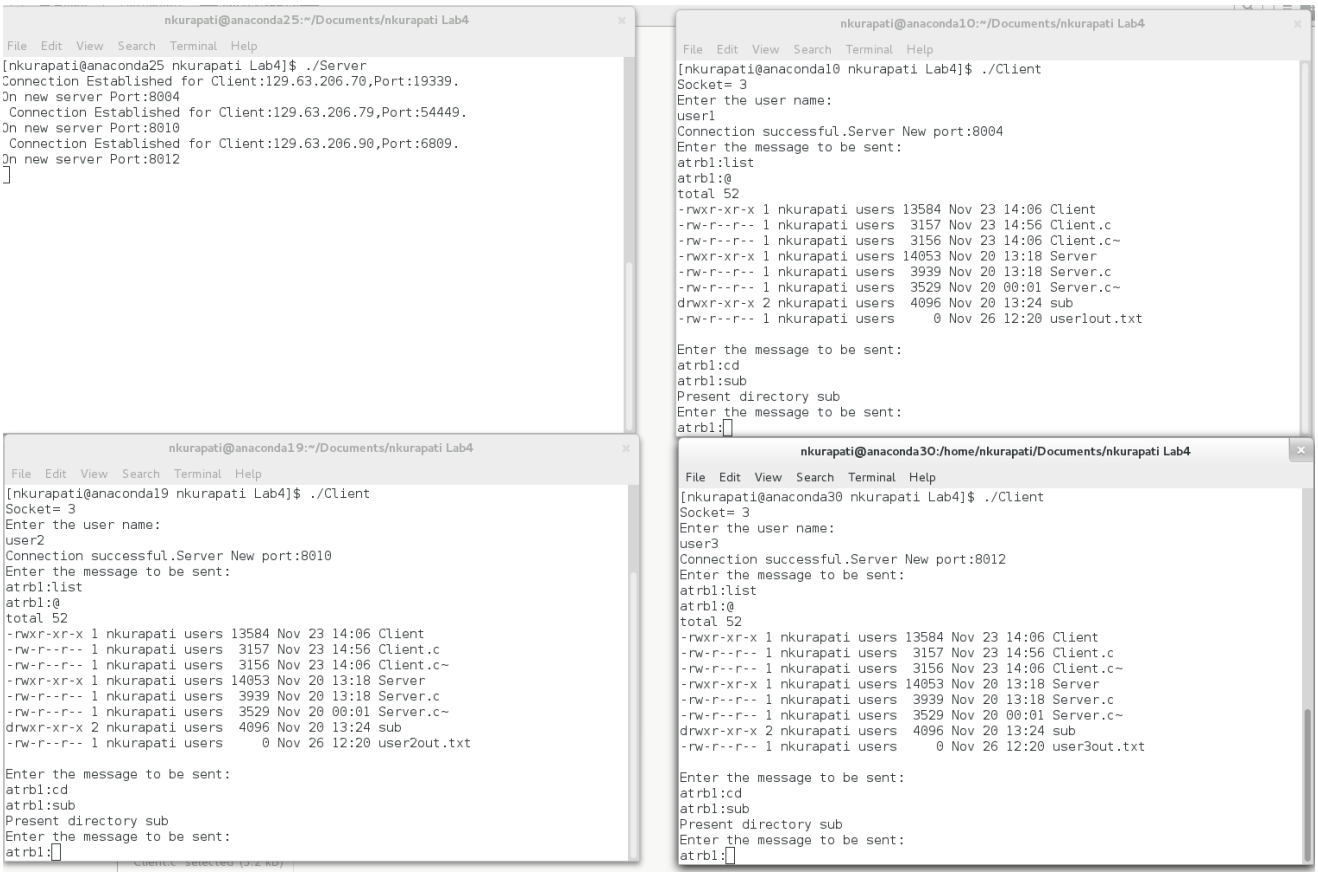

Fig.3 – Demonstrating the cd command

You can see in the fig. when the first attribute is cd and second is the sub folder name, then it changes the subdirectory and send the reply.

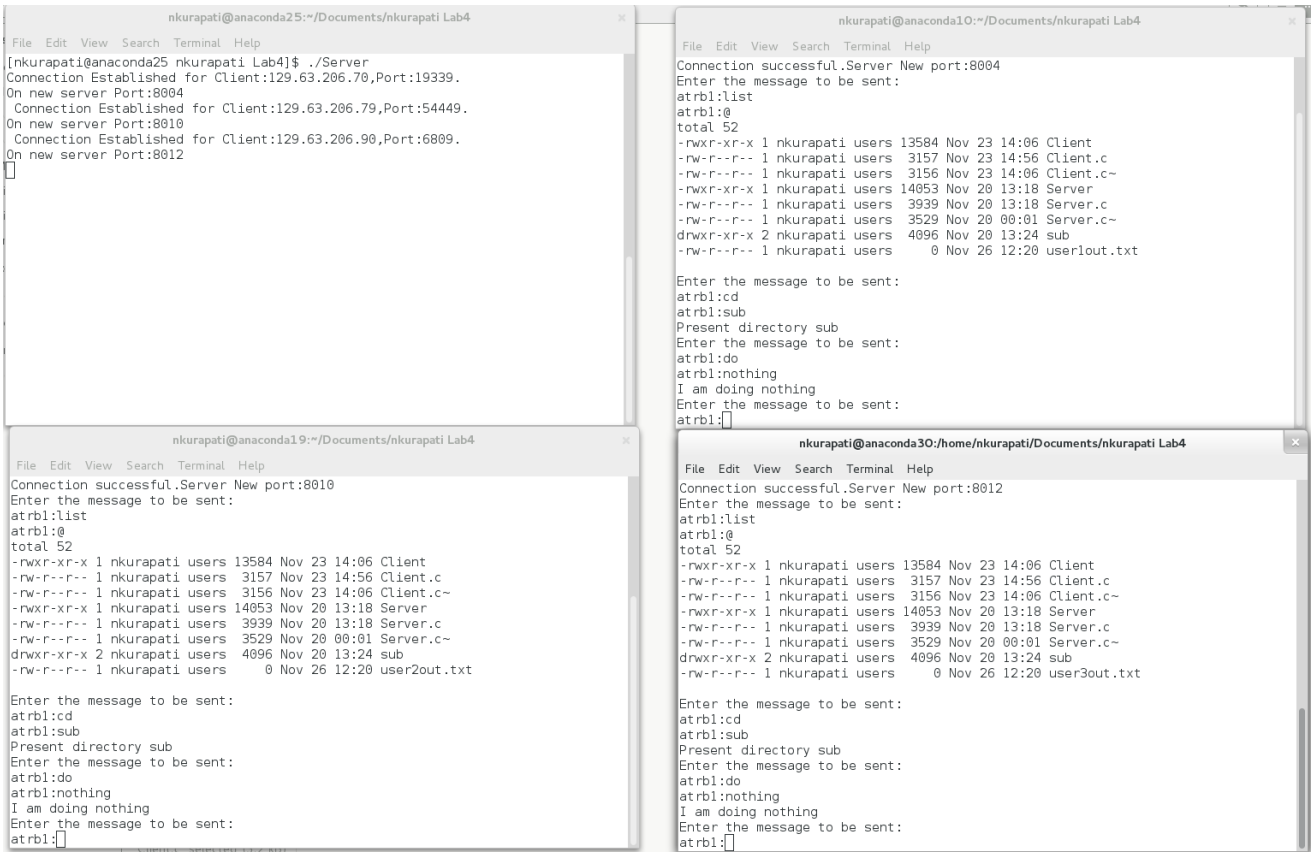

Fig.4 – Demonstrating do nothing command

You can see the above fig. first attribute is “do”, second is “nothing”. Server reply is “I am doing nothing”

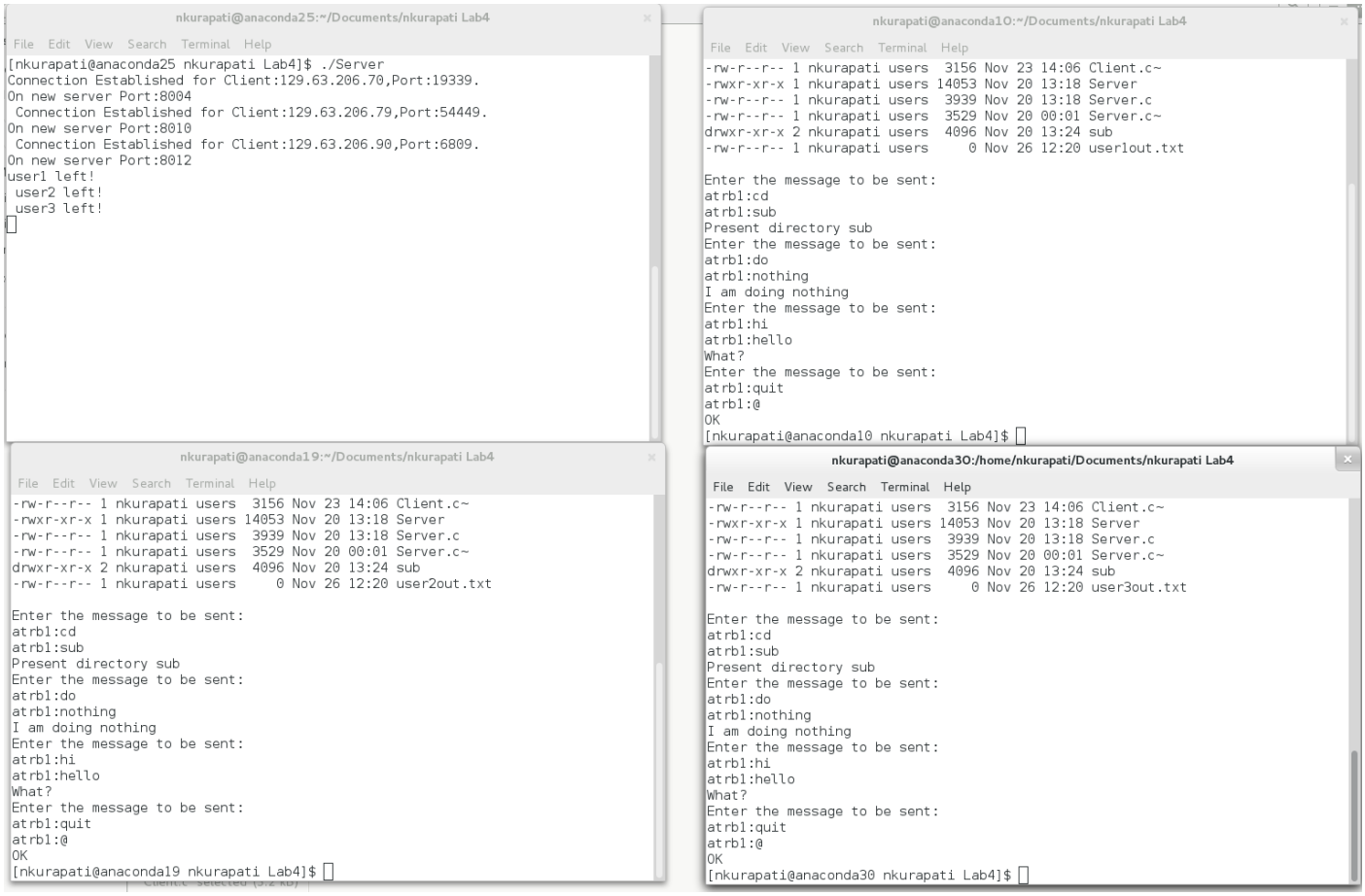

Fig.5- Demonstrating other unknown commands and quit command

You can see in the above fig., when unknown command is send to it, server replies “what?”. When it quit command is send it reply OK.

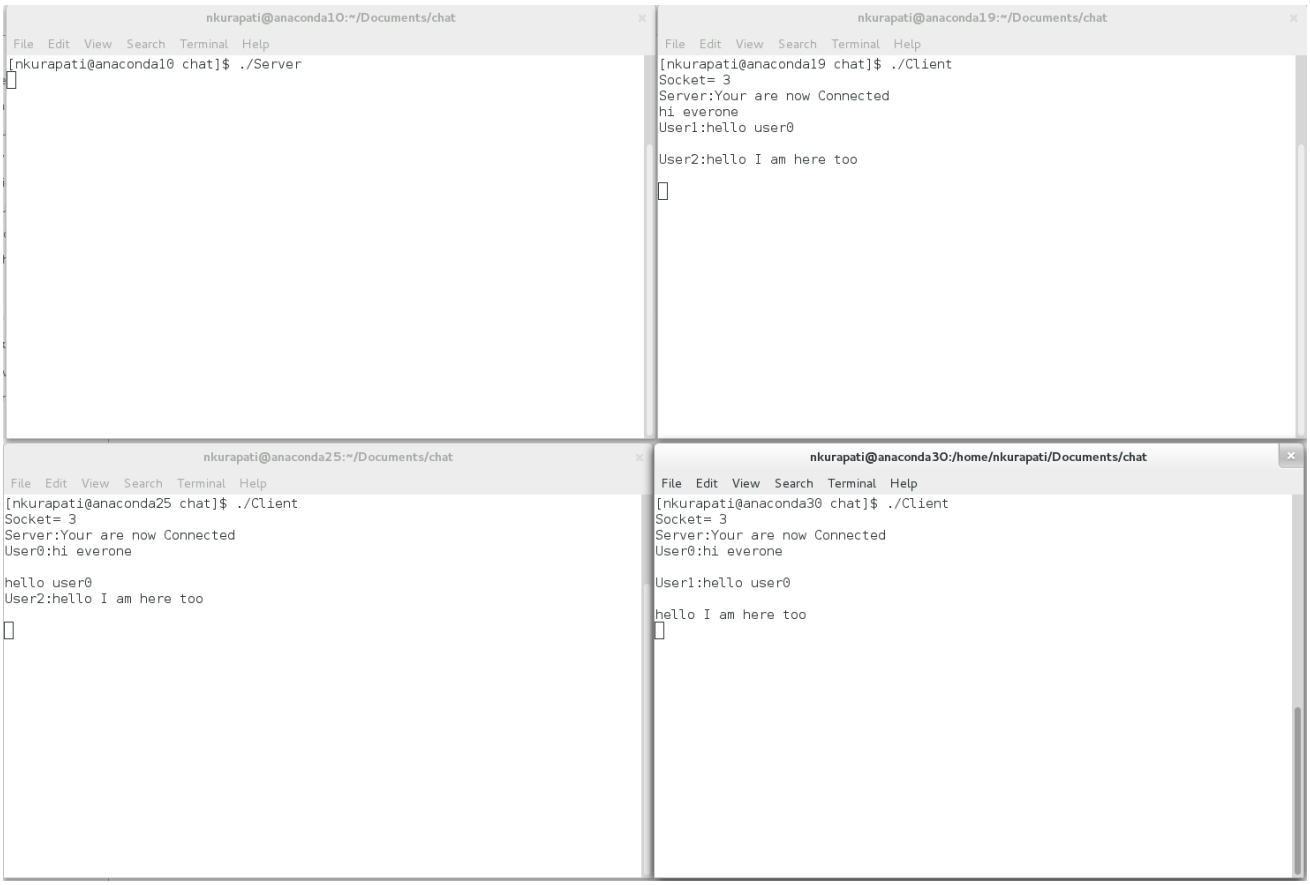

Fig.6 – Demonstrating chat application

If you see the above fig., it shows the working of chat application. First the server is running on anaconda10, 3 clients on anaconda19,25,30. Once the client application is executed, it establishes the connection to the server automatically.

5) Observations:

For the concurrent operation of the server, it need to use different processes to handle the individual clients. Each process need have different socket with unique port number to maintain the connection to the client. Server listen on the well-known port to initiate the connection. When it receives the client request, it fort new process to individual client and assign unique port number. The new process will service the client.

When it comes to chat application, on the client side, it as to receive the messages from the chat room every time. This can be achieved by using pthreads to parallelize the receive and send operation over the socket. On the server side, it need to receive the message from the clients and send them to everyone. Since it stores IP addresses of each client during connection initiation. Then it spans a group of threads to send the message to all the clients in the chat room. We can say sockets are thread safe, so no need of using mutex or semaphores.

6) Conclusions:

The remote procedure on server can be called from the client over the networking using sockets. The command is executed on the server using system calls, and directed output to a text file. Then it is read in to a buffer and send it to client. Then client prints the output on the screen.

7) Source Code:

See the attachment to find the source code of Client, Server of Project and Client, Server of chat application respectively.

```c
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<strings.h>
#include<string.h>
#include<unistd.h>
#include<stdbool.h>
#define size 1024
/* Server machine */
/* Declaring errno */
extern int errno;

/* Function for printing error */
void report_error(char *s)
{
  printf("receiver: error in%s, errno = %d\n", s, errno);
  exit(1);
}

//To intialize given socket
void intializeSocket(int *s, struct sockaddr_in *sa,int portNo,int backlog)
{
  /* Creating the socket and returns error if unsuccesfull */
  if((*s= socket(AF_INET, SOCK_DGRAM, PF_UNSPEC)) == -1)
    report_error("socket");
  sa->sin_family = AF_INET;
  sa->sin_addr.s_addr=INADDR_ANY;
  sa->sin_port = htons(portNo); /* define port number based on student ID*/
  /* Binding the socket and returns error if unsuccesfull */
  if(bind(*s, (struct sockaddr *)sa, sizeof(*sa))== -1)
    report_error("bind");
  listen(*s, backlog);
}

//To get request from the client
void GetRequest(char *msg, int s, struct sockaddr_in *r_sa)
{
  int r_sa_l = sizeof(*r_sa);
  /* Receiving message from client and returns error if unsuccessfull */
  if(recvfrom(s, msg, size, 0, (struct sockaddr *)r_sa, &r_sa_l)<0)
    report_error("child recvfrom");
}

//To send reply to the client
void SendReply(char *response, int s, struct sockaddr_in *r_sa)
{
  int r_sa_l = sizeof(*r_sa);
  /* Sending response to client */
  if(sendto(s,response,size,0,(struct sockaddr *)r_sa,r_sa_l)<0)
    report_error("child sendto");
}

/* Dynamically giving the 'size' of message as argument */
void main(int argc, char *argv[])
{
  int s;
  int childNo = 0;
  struct sockaddr_in sa = {0}, r_sa = {0};
  char username[size];
  char response[size];
  char msg[size];
  //Initailize the socket
  intializeSocket(&s, &sa,4079,10);
```

```
pid_t pid;
while(1)
  {
    GetRequest(msg,s,&r_sa);
    strcpy(username,msg);
    //Fork a child process
    pid = fork();
    //fork fails
    if(pid<0)
      {
        report_error("fork");
        exit(1);
      }
    //child process
    if(pid == 0)
      {
        close(s);
        int cs;
        struct sockaddr_in csa = {0};
        //Initialize the new socket with new port number
        intializeSocket(&cs, &csa,2000+getpid(),1);
        //Buffers to read two attributes
        char msgBuf1[10],msgBuf2[10];
        bool breakloop = false;
        while(1)
          {
            //Flush the buffer
            bzero(response,size);
            GetRequest(msg,cs,&r_sa);
            // printf("%s: %s\n",username,msg);
            sscanf(msg,"%s %s",msgBuf1,msgBuf2);
            //printf("%s,%s\n",msgBuf1,msgBuf2);
            //Execute list command
            if(strcmp(msgBuf1,"list")==0)
              {
                char listBuf[1024];
                char fileName[20];
                sprintf(fileName,"%sout.txt",username);
                sprintf(listBuf,"ls -l > %s",fileName);
                system(listBuf);
                FILE * fp= fopen(fileName, "rb");
                fseek(fp,0,SEEK_END);
                long filesize = ftell(fp);
                fseek(fp,0,SEEK_SET);
                fread(response,filesize,1,fp);
                fclose(fp);
                sprintf(listBuf,"rm %s",fileName);
                system(listBuf);

              }
            //Execute 'cd' command
            else if(strcmp(msgBuf1,"cd")==0)
              {
                if(chdir(msgBuf2)<0)
                  strcpy(response,"No such directory");
                else
                  sprintf(response,"Present directory %s",msgBuf2);
              }
            //Do noting
            else if(strcmp(msgBuf1,"do")==0)
              {
                strcpy(response,"I am doing nothing");
              }
            //Quit command
            else if(strcmp(msgBuf1,"quit")==0)
              {
                strcpy(response,"OK");
```

```
                breakloop = true;
              }
            else
              {
                strcpy(response,"What?");
              }
            //Send the response
            SendReply(response,cs,&r_sa);
            if(breakloop)
              break;
          }
        printf("%s left!\n",username);
        close(cs);
        exit(0);
      }
    else //server part
      {
        childNo++;
        int childPortNo = 2000+pid;
        //Send the new port assigned
        sprintf(response,"OK %d",childPortNo);
        SendReply(response,s,&r_sa);
        char ipBuffer[20];
        inet_ntop(AF_INET, &(r_sa.sin_addr), ipBuffer, 20);
        printf("Connection Established for Client:%s,Port:%d. \nOn new server Port:%d
\n",ipBuffer,r_sa.sin_port,childPortNo);
      }
  }
 //Close the socket and wait for the child process to complete
 if(pid!=0)
  {
    close(s);
    int i;
    for(i=0;i<childNo;i++)
      wait(NULL);
  }
}
```

```
/#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/time.h>
#include<errno.h>
#include<arpa/inet.h>
#include<string.h>
#include<stdbool.h>
#define RECEIVER_HOST "anaconda25.uml.edu" /* Server machine */
#define BUFSIZE 1024
/* Declaring errno */
extern int errno;
/* Function for error */
void report_error(char *s)
{
  printf("sender: error in %s, errno = %d\n",s,errno);
  exit(1);
}

//To Send and receive from the server
bool DoOperation(char *msg, char *received, int s, struct sockaddr_in sa)
{
  int length = sizeof(sa);
  /* Sending the message to server and returns error if unsuccesfull */
  if(sendto(s, msg, BUFSIZE, 0, (struct sockaddr *) &sa, length)== -1)
    report_error("sendto");
  struct timeval tTmp;
  tTmp.tv_sec = 3;
  tTmp.tv_usec = 0;
  if(setsockopt(s, SOL_SOCKET, SO_RCVTIMEO,&tTmp,sizeof(tTmp)) < 0)
    report_error("timeout");
  /* Receives message from server and returns error if unsuccesfull */
  if(recvfrom(s, received, BUFSIZE, 0, (struct sockaddr *) &sa, &length)<0)
    return false;
  return true;
}

/* Giving 'size' of message dynamically as argument */
void main(int argc, char *argv[])
{
  int s,i;
  char msg[BUFSIZE];
  char received[BUFSIZE];
  struct hostent *hp;
  struct sockaddr_in sa= {0};

  /* FILL SOCKET ADDRESS*/
  if((hp = gethostbyname(RECEIVER_HOST))==NULL)
    report_error("gethostbyname");
  bcopy((char*)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
  sa.sin_family = hp->h_addrtype;
  sa.sin_port = htons(4079); /* define port number based on student ID*/

  /* Creating the socket and returns error if unsuccessfull */
  if((s=socket(AF_INET, SOCK_DGRAM, PF_UNSPEC))== -1)
    report_error("socket");
  printf("Socket= %d\n",s);

  bzero(msg, BUFSIZE);
  bzero(received, BUFSIZE);
  //Connecting to the server
  printf("Enter the user name: \n");
  scanf("%s",msg);
```

```
  i =0;
  for(i=0;i<3;i++)
    {
      //To get new port after sending the username
      if(DoOperation(msg,received,s,sa))
        {
          char recBuf[2];
          sscanf(received,"%s",recBuf);
          int SerportNo;
          //Extracting the new port from the message received
          sscanf(received,"%*[^0-9]%d",&SerportNo);
          //printf("%s\n",recBuf);
          //printf("%d\n", SerportNo);
          if(strcmp(recBuf,"OK")!=0)
            {
              printf("New port Not received \n");
              return;
            }
          //Updating the server new port
          sa.sin_port = htons(SerportNo);
          printf("Connection successful.Server New port:%d \n", SerportNo);
          // printf("%s\n",received);
          break;
        }
      else
        {
          if(i==2)
            {
              printf("Connection Unsuccessful \n");
              return;
            }
        }
    }

  while(1)
    {
      //Flush the buffers
      bzero(msg, BUFSIZE);
      bzero(received, BUFSIZE);
      printf("Enter the message to be sent: \n");
      char msg1[10],msg2[10];
      //Get the user input, two attributes
      printf("atrb1:");
      scanf("%s",msg1);
      printf("atrb2:");
      scanf("%s",msg2);
      sprintf(msg,"%s %s",msg1,msg2);
      // printf("%s\n",msg);
      i =0;
      //Send and receive the message
      for(i=0;i<3;i++)
        {
          if(DoOperation(msg,received,s,sa))
            {
              printf("%s\n",received);
              break;
            }
        }
      if(i==3)
        printf("Server Not Reachable.\n");
      //Quit on OK message
      if((strcmp(received,"OK")==0)&&(strcmp(msg1,"quit")==0))
        break;
    }
  close(s);
}
```

```c
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
#include<errno.h>
#include<strings.h>
#include<string.h>
#include<unistd.h>
#include<stdbool.h>
#include<pthread.h>
#include<semaphore.h>
#include<signal.h>

#define size 1024
#define maxClientNo 10
/* Server machine */
/* Declaring errno */
extern int errno;

//globals
int s;
char msg[size];
char response[size];
char username[20];
struct sockaddr_in rsa[maxClientNo] = {{0}};//Ip buffer
struct sockaddr_in sa = {0}, r_sa ={0};

/* Function for printing error */
void report_error(char *s)
{
  printf("receiver: error in%s, errno = %d\n", s, errno);
  exit(1);
}

//To intialize given socket
void intializeSocket(int *s, struct sockaddr_in *sa,int portNo,int backlog)
{
  /* Creating the socket and returns error if unsuccesfull */
  if((*s= socket(AF_INET, SOCK_DGRAM, PF_UNSPEC)) == -1)
    report_error("socket");
  sa->sin_family = AF_INET;
  sa->sin_addr.s_addr=INADDR_ANY;
  sa->sin_port = htons(portNo); /* define port number based on student ID*/
  /* Binding the socket and returns error if unsuccesfull */
  if(bind(*s, (struct sockaddr *)sa, sizeof(*sa))== -1)
    report_error("bind");
  listen(*s, backlog);
}

//To get request from the client
void GetRequest()
{
  int r_sa_l = sizeof(r_sa);
  /* Receiving message from client and returns error if unsuccessfull */
  if(recvfrom(s, msg, size, 0, (struct sockaddr *)&r_sa, &r_sa_l)<0)
    report_error("child recvfrom");
}

//To send reply to the client
void SendReplyServer()
{
  int r_sa_l = sizeof(r_sa);
  /* Sending response to client */
  if(sendto(s,response,size,0,(struct sockaddr *)&r_sa,r_sa_l)<0)
    report_error("server sendto");
```

```
}

//To send reply to the client
void *SendReply(void* data)
{
  struct sockaddr_in local_rsa =*((struct sockaddr_in *)data);
  int r_sa_l = sizeof(local_rsa);
  /* Sending response to client */
  if(sendto(s,response,size,0,(struct sockaddr *)&local_rsa,r_sa_l)<0)
    report_error("child sendto");
}


/* Dynamically giving the 'size' of message as argument */
void main(int argc, char *argv[])
{
  int i;
  bool flag = false;
  pthread_t pClient[maxClientNo];
  int childNo = 0;
  //Intialize the socket
  intializeSocket(&s, &sa,4079,50);
  while(1)
    {
      bzero(response,size);
      //Get the client response
      GetRequest(msg,s,&r_sa);
      //Initial connection
      if(strcmp(msg,"connect")==0)
        {
          //Search the IP adress array to avoid connection clash
          for(i=0;i<childNo;i++)
            {
              if(r_sa.sin_addr.s_addr == rsa[i].sin_addr.s_addr)
                {
                  if(r_sa.sin_port == rsa[i].sin_port)
                    {
                      sprintf(response,"Server:Your are already Connected");
                      SendReplyServer();
                      flag = true;
                    }
                }
            }
          if(flag)
            {
              flag = false;
              continue;
            }
          //Accept new connection
          if(childNo<maxClientNo)
            {
              rsa[childNo] = r_sa;
              childNo++;
              sprintf(response,"Server:Your are now Connected");
              SendReplyServer();
              continue;
            }
          //If IP buffer is full
          else
            {
              sprintf(response,"Server:Chatroom full");
              SendReplyServer();
              continue;
            }

        }
      //If only one user in the chat room
```

```
      if(childNo==1)
        {
          sprintf(response,"Server:No one in the chat room");
          SendReplyServer(response,s,&r_sa);
          continue;
        }
      bzero(username,20);
      //printf("Childno:%d\n",childNo);
      int k;
      //Scan the IP buffer to find who send the message
      for(k=0;k<childNo;k++)
        {
          if(r_sa.sin_addr.s_addr == rsa[k].sin_addr.s_addr)
            {
              if(r_sa.sin_port == rsa[k].sin_port)
                {
                  sprintf(username,"User%d",k);
                  break;
                }
            }
        }
      bzero(response,size);
      sprintf(response,"%s:%s",username,msg);
      long j=0;
      //Create service threads
      for(j=0;j<childNo;j++)
        {
          if(j!=k)
          pthread_create(&pClient[j], NULL, SendReply, (void*) &rsa[j]);
        }
      //Wait for Producer and consumer threads to finish
      for(j=0;j<childNo;j++)
        {
          if(j!=k)
          pthread_join(pClient[j], NULL);
        }
    }
  close(s);
}
```

```c
#include<sys/types.h>
#include<sys/socket.h>
#include<netdb.h>
#include<netinet/in.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/socket.h>
#include<sys/time.h>
#include<errno.h>
#include<arpa/inet.h>
#include<string.h>
#include<stdbool.h>
#define RECEIVER_HOST "anaconda10.uml.edu" /* Server machine */
#define BUFSIZE 1024
/* Declaring errno */
extern int errno;

//Globals
int s;
char msg[BUFSIZE];
char received[BUFSIZE];
struct sockaddr_in sa= {0};

/* Function for error */
void report_error(char *s)
{
  printf("sender: error in %s, errno = %d\n",s,errno);
  exit(1);
}

//Reply function for thread
void SendReply()
{
  int length = sizeof(sa);
  /* Sending the message to server and returns error if unsuccesfull */
  if(sendto(s, msg, BUFSIZE, 0, (struct sockaddr *) &sa, length)== -1)
    report_error("sendto");

}

//Get function
void *GetRequest()
{
  int length = sizeof(sa);
  while(1)
    {
       bzero(received, BUFSIZE);
      /* Receives message from server and returns error if unsuccesfull */
      if(recvfrom(s, received, BUFSIZE, 0, (struct sockaddr *) &sa, &length)<0)
        report_error("Requestfrom");
      printf("%s\n",received);
    }
}

/* Giving 'size' of message dynamically as argument */
void main(int argc, char *argv[])
{
  int i;
  struct hostent *hp;

  /* FILL SOCKET ADDRESS*/
  if((hp = gethostbyname(RECEIVER_HOST))==NULL)
    report_error("gethostbyname");
  bcopy((char*)hp->h_addr, (char *)&sa.sin_addr, hp->h_length);
  sa.sin_family = hp->h_addrtype;
  sa.sin_port = htons(4079); /* define port number based on student ID*/
```

```
  /* Creating the socket and returns error if unsuccessfull */
  if((s=socket(AF_INET, SOCK_DGRAM, PF_UNSPEC))== -1)
    report_error("socket");
  printf("Socket= %d\n",s);
  pthread_t p;
  //Create a thread to receive the messages from the server
  pthread_create(&p,NULL,GetRequest,NULL);
  //To connect to the server
  sprintf(msg,"connect");
  SendReply();
  while(1)
    {
      bzero(msg, BUFSIZE);
      // printf("Enter the message to be sent: \n");
      // scanf("%s",msg);
      //Scan the message from user and send
      fgets(msg,sizeof(msg),stdin);
      SendReply();
    }
  pthread_join(p,NULL);
  close(s);
}
```