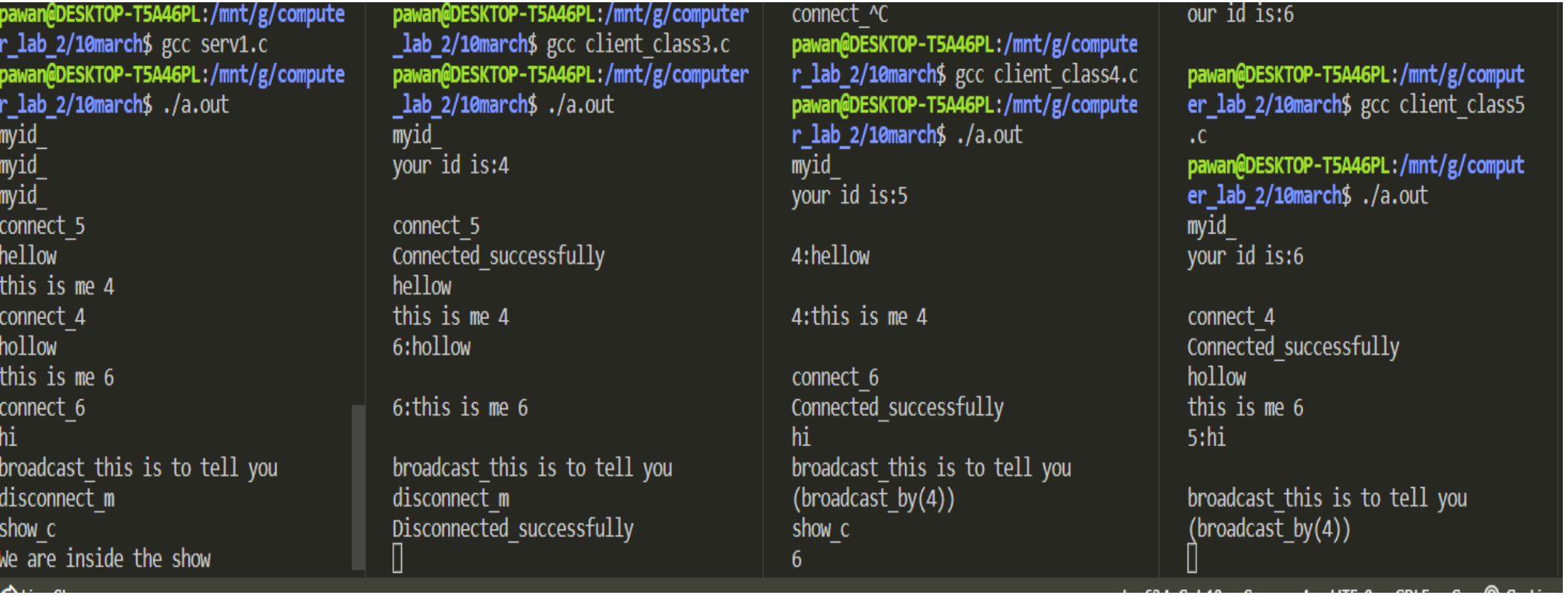
The intent of this report is make one understand the vivid and clear working of the program.

Below is the attached snapshot of porgram.



serv1.c client_class3.c client_class4.c client_class5.c

now the running is explained

“myid_” : this statement (without quotes) tell the id to the user.

“connect_5” :this statement(without quotes) connects the present user to other user

. with id 5 once it is connected to you can talk to the other unless

. you type “disconnect_m”

“disconnect_m” : this statement (without quotes) makes the user offline.

“broadcast_hellow” : this statement(without quotes) would “broadcast” hellow to all

. active clients.

“show_c” :this statement (without quotes) will show all the active clients.

The above commands are made for user friendly purpose.

“/makegroup <client id1> <client id2> ... <client idn>” : A group with unique id will be made including all the mentioned clients along with the admin client.

“ /makegroupreq <client id1> <client id2> ... <client idn>” : A group having unique id should be made with currently only the admin client.

. The request message for joining the group should be notified to all the specified clients.

. Clients can respond to join that group.

“/joingroup <group id>” : If this message is sent by a client having the request joining the group, then he will be added to the group immediately.

.

“ /declinegroup <group id>” : If this message is sent by a client having the request for

joining the group, then the client will not be added to
. the group.

“ /sendgroup <group id> <Message>” :The sender should be in the group to transfer the
message to all his peers of that group. The
. message should be sent to all the peers
along with group info.

“ /makeadmin <group id> <client id>” : The admin should be able to alleviate the
. privileges of client id to that of admin.

“ /addtogroup <group id> <client id1> <client id2> ... <client idn> “: The admin should
be able to add member(s) to the group.

“ /removefromgroup <group id> <client id1> <client id2> ... <client idn>” : The admin
should be able to remove member(s) from the group.

“/makegroupbroadcast <group id>” :Any admin of the group should be able to modify
the type of group as broadcast-only in which only
. admins are allowed to message.

“/activegroups” : To display all the groups that are currently active on the
. and the sender is a part of. Here you have to display at the .
. client side the group ids followed by

the group admin’s client id, and the ids of all the clients that
. are part of this group.

“ /quit” : The client will be removed from the server. This client will be
. removed from all the active groups

Explanation in simpler words;

```
struct grp
{
    int bonly;              // for any particular grp this means if bonly is set one the group is broadcast only.
    int used;               // this means the group is free to be used if set to zero. Otherwise not
    int members[100005];     // if members[i] is set one means that member is part of group.
    int admins[100005];           // if admins[i] is set one means that admins[i] is admin of group.

} group[100];                    // there are 100 possible groups.

int option_is_valid[100][100005];       // if options[i][j] is set one means group i’s supposedly client j has
//                                        not responded yes to choice of being selected in a group. Otherwise //
..                                       yes if zero.
```

We have used clients in file discriptor and used them as.

```
for(int i=0;i<FD_SETSIZE;i++)
     {
         if(FD_ISSET(i,&ready_sockets))
         {
             // printf("We are working on server:%d\n",i);
             if(i==server_socket)                            // new socket.
             {
                 //new connection
                 int client_socket=accept_new_connection(server_socket);  // the new connection is here.

                 FD_SET(client_socket,&current_sockets);         // new client is addeded to filedescriptor.
                 available_clients[client_socket]=1;             // set available client to one.

             }
             else
             {
                 handle_content(i);                          // handle content if already spawned clinet.
                 // FD_CLR(i,&current_sockets);
             }
         }
     }
```

All the rest functions have been accordingly traversed as per the demands from user by manipulation of above data structure.

```
 if(buffer[0]=='m'&&buffer[1]=='y'&&buffer[2]=='i'&&buffer[3]=='d'&&buffer[4]=='_')
      {

         char idno[256];
         char mssg[256];
         sprintf(idno,"%d",client_socket);
         strcpy(mssg,"your id is:");
         strcat(mssg,idno);
         strcat(mssg,"\n");
         n=write(client_socket,mssg,sizeof(mssg));
         bzero(buffer,256);
         return NULL;

      }

      if(buffer[0]=='/'&&buffer[1]=='m'&&buffer[2]=='a'&&buffer[3]=='k'&&buffer[4]=='e'&&buffer[5]=='g'&&buffer[
6]=='r'&&buffer[7]=='o'&&buffer[8]=='u'&&buffer[9]=='p'&&buffer[10]==' ')
       {
        int  grpid;
        int  av_flg=0;

        for(int i=0;i<100;i++)
        {
             if(group[i].used==0)
             {
                grpid=i;
                group[i].used=1;
                av_flg=1;
```

```
            break;
         }
    }

    if(av_flg==0)
    {
        char not_msg[256];
        strcpy(not_msg,"sorry could not create a group\n");
        n=write(client_socket,not_msg,sizeof(not_msg));
        bzero(buffer,256);

        return NULL;


                // send message to requesting client that the group can't be formed
                return NULL;
    }

    group[grpid].admins[client_socket]=1;
    group[grpid].members[client_socket]=1;



    int val=0;
    for(int i=10;buffer[i]!='\n';)
    {
        if(buffer[i]==' ')
        {
            i+=1;
            val=0;
            while(buffer[i]!=' '&&buffer[i]!='\n')
            {
                val=val*10+(buffer[i]-'0');
```

```
                    i+=1;
                }
                group[grpid].members[val]=1;      // add members to the groupid;

                if(buffer[i]=='\n')
                    break;
            }
        }

        char id_char[256];
        char grp_msg[256];
        strcpy(grp_msg,"group is made suffessfully with group id=");
        sprintf(id_char,"%d",grpid);
        strcat(grp_msg,id_char);
        strcat(grp_msg,"\n");
        n=write(client_socket,grp_msg,sizeof(grp_msg));
        bzero(buffer,256);

        return NULL;




       }


      if(buffer[0]=='/'&&buffer[1]=='m'&&buffer[2]=='a'&&buffer[3]=='k'&&buffer[4]=='e'&&buffer[5]=='g'&&buffer[
6]=='r'&&buffer[7]=='o'&&buffer[8]=='u'&&buffer[9]=='p'&&buffer[10]=='r'&&buffer[11]=='e'&&buffer[12]=='q'&&buff
er[13]==' ')
       {

        int  grpid;
        int  av_flg=0;
```

```
for(int i=0;i<100;i++)
{
     if(group[i].used==0)
     {
        grpid=i;
        group[i].used=1;
        av_flg=1;
        break;
     }
}

for(int i=0;i<100005;i++)
 option_is_valid[grpid][i]=1;

if(av_flg==0)
{
    char not_msg[256];
    strcpy(not_msg,"sorry could not create a group\n");
    n=write(client_socket,not_msg,sizeof(not_msg));
    bzero(buffer,256);

    return NULL;


            // send message to requesting client that the group can't be formed
            return NULL;
}

group[grpid].admins[client_socket]=1;
group[grpid].members[client_socket]=1;
```

```
        int val=0;
        for(int i=13;buffer[i]!='\n';)
        {
            if(buffer[i]==' ')
            {
                i+=1;
                val=0;
                while(buffer[i]!=' '&&buffer[i]!='\n')
                {
                    val=val*10+(buffer[i]-'0');
                    i+=1;
                }
                group[grpid].members[val]=0;      // add members to the groupid;
                char mssg[256];
                char grp_char[256];
                sprintf(grp_char,"%d",grpid);
                strcpy(mssg,"would you like to join the group ");
                strcat(mssg,grp_char);
                strcat(mssg," (your choice write /joingroup <group id> or /declinegroup <group id>");
                n=write(val,mssg,sizeof(mssg));
                bzero(mssg,256);
                // return NULL;

                if(buffer[i]=='\n')
                    break;
            }
        }

        char id_char[256];
        char grp_msg[256];
        strcpy(grp_msg,"group is made suffessfully with group id=");
        sprintf(id_char,"%d",grpid);
```

```
        strcat(grp_msg,id_char);
        strcat(grp_msg,"\n");
        n=write(client_socket,grp_msg,sizeof(grp_msg));
        bzero(buffer,256);

        return NULL;




      }

// 3. /joingroup <group id> : If this message is sent by a client having the request for joining
// the group, then he will be added to the group immediately

 if(buffer[0]=='/'&&buffer[1]=='j'&&buffer[2]=='o'&&buffer[3]=='i'&&buffer[4]=='n'&&buffer[5]=='g'&&buffer[6]=='
r'&&buffer[7]=='o'&&buffer[8]=='u'&&buffer[9]=='p'&&buffer[10]==' ')
 {

    int grpid=0;
    int i=11;
    int val=0;
    for(;buffer[i]!=' '&&buffer[i]!='\n';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    grpid=val;

    if(option_is_valid[grpid][client_socket]==0)
    {
    char mssg[256];
    strcpy(mssg,"You have already responded!\n");
    n=write(client_socket,mssg,sizeof(mssg));
```

```
    bzero(buffer,256);
    return NULL;

    }
    option_is_valid[grpid][client_socket]=0;


    group[grpid].members[client_socket]=1;
    char mssg[256];
    strcpy(mssg,"You are successfully added to the group\n");
    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;


 }

  if(buffer[0]=='/'&&buffer[1]=='d'&&buffer[2]=='e'&&buffer[3]=='c'&&buffer[4]=='l'&&buffer[5]=='i'&&buffer[6]==
'n'&&buffer[7]=='e'&&buffer[8]=='g'&&buffer[9]=='r'&&buffer[10]=='o'&&buffer[11]=='u'&&buffer[12]=='p'&&buffer[1
3]==' ')
 {
    int grpid=0;
    int i=14;
    int val=0;
    for(;buffer[i]!=' '&&buffer[i]!='\n';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    grpid=val;

    if(option_is_valid[grpid][client_socket]==0)
    {
    char mssg[256];
```

```
    strcpy(mssg,"You have already responded!\n");
    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;

    }
    option_is_valid[grpid][client_socket]=0;

    group[grpid].members[client_socket]=0;
    char mssg[256];
    strcpy(mssg,"You are not added to the group thanks for responding\n");
    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;


 }









// /sendgroup <group id> <Message>: The sender should be in the group to transfer the
// message to all his peers of that group. The message should be sent to all the peers
// along with group info.

 if(buffer[0]=='/'&&buffer[1]=='s'&&buffer[2]=='e'&&buffer[3]=='n'&&buffer[4]=='d'&&buffer[5]=='g'&&buffer[6]=='
r'&&buffer[7]=='o'&&buffer[8]=='u'&&buffer[9]=='p'&&buffer[10]==' ')
 {
```

```
    int grpid=0;
    int i=11;
    int val=0;
    for(;buffer[i]!=' ';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    grpid=val;
    if(group[grpid].bonly==1&&group[grpid].admins[client_socket]==0)
    {
        char mssg[256];
        strcpy(mssg,"Sorry !Only admins can send message");
        n=write(client_socket,mssg,sizeof(mssg));
        bzero(buffer,256);
        return NULL;
    }
    i+=1;
    val=0;
    char grpmsg[256];
    int k=0;
    for(;buffer[i]!='\n';i++)
    {
        grpmsg[k]=buffer[i];
        k+=1;
    }
    grpmsg[k]='\0';
    char id_char[256];
    char grp_msg[256];
    char grpid_char[256];
    strcpy(grp_msg,"(sent by client ");
    sprintf(id_char,"%d",client_socket);
    sprintf(grpid_char,"%d",grpid);
```

```
    strcat(grp_msg,id_char);
    strcat(grp_msg," from group ");
    strcat(grp_msg,grpid_char);
    strcat(grp_msg," )");
    strcat(grpmsg,grp_msg);
    strcat(grpmsg,"\n");
    // n=write(client_socket,grp_msg,sizeof(grp_msg));
    // bzero(buffer,256);



    if(group[grpid].members[client_socket]==1)
    for(int i=0;i<100005;i++)
    {
        if(group[grpid].members[i]==1&&i!=client_socket&&available_clients[i]==1)
        {

            n=write(i,grpmsg,sizeof(grpmsg));
            bzero(buffer,256);
        }

    }

    bzero(buffer,256);
    return NULL;


 }
//makeadmin
 if(buffer[0]=='/'&&buffer[1]=='m'&&buffer[2]=='a'&&buffer[3]=='k'&&buffer[4]=='e'&&buffer[5]=='a'&&buffer[6]=='
d'&&buffer[7]=='m'&&buffer[8]=='i'&&buffer[9]=='n'&&buffer[10]==' ')
{
```

```
    int grpid=0;
    int i=11;
    int val=0;
    for(;buffer[i]!=' ';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    grpid=val;

    if(group[grpid].admins[client_socket]==0)
    {
        char mssg[256];
        strcpy(mssg,"Sorry you are not admin of the group!");
        n=write(client_socket,mssg,sizeof(mssg));
        bzero(buffer,256);
        return NULL;
    }

    i+=1;
    val=0;
    for(;buffer[i]!='\n';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    int to_elevate=val;
    group[grpid].admins[to_elevate]=1;

    char id_char[256];
    char grp_msg[256];
    strcpy(grp_msg,"cleint is added to the group admin with group id=");
    sprintf(id_char,"%d",to_elevate);
    strcat(grp_msg,id_char);
```

```
    strcat(grp_msg,"\n");
    n=write(client_socket,grp_msg,sizeof(grp_msg));
    bzero(buffer,256);


    return NULL;

}

//  /addtogroup <group id> <client id1> <client id2> ... <client idn> : The admin should
// be able to add member(s) to the group.

if(buffer[0]=='/'&&buffer[1]=='a'&&buffer[2]=='d'&&buffer[3]=='d'&&buffer[4]=='t'&&buffer[5]=='o'&&buffer[6]=='g
'&&buffer[7]=='r'&&buffer[8]=='o'&&buffer[9]=='u'&&buffer[10]=='p'&&buffer[11]==' ')
{
      int  grpid;
        int  av_flg=0;



        // for(int i=0;i<100;i++)
        // {
        //      if(group[i].used==0)
        //      {
        //         grpid=i;
        //         group[i].used=1;
        //         av_flg=1;
        //         break;
        //      }
        // }

        // if(av_flg==0)
```

```
        // {
        //     char not_msg[256];
        //     strcpy(not_msg,"sorry could not create a group\n");
        //     n=write(client_socket,not_msg,sizeof(not_msg));
        //     bzero(buffer,256);

        //     return NULL;


        //             // send message to requesting client that the group can't be formed
        //             return NULL;
        // }

        // group[grpid].admins[client_socket]=1;
        // group[grpid].members[client_socket]=1;


        int count=0;
        int val=0;
        for(int i=11;buffer[i]!='\n';)
        {
            if(buffer[i]==' ')
            {
                i+=1;
                val=0;
                while(buffer[i]!=' '&&buffer[i]!='\n')
                {
                    val=val*10+(buffer[i]-'0');
                    i+=1;
                }
                if(count==0)
                { count+=1;
                  grpid=val;
```

```
                if(group[grpid].admins[client_socket]==0)
                  {
                       char mssg[256];
                      strcpy(mssg,"Sorry you are not admin of the group!");
                       n=write(client_socket,mssg,sizeof(mssg));
                       bzero(buffer,256);
                      return NULL;
                  }
              }
               else
              group[grpid].members[val]=1;      // add members to the groupid;

              if(buffer[i]=='\n')
                  break;
          }
      }

      char id_char[256];
      char grp_msg[256];
      strcpy(grp_msg,"cleints are added to the group with group id=");
      sprintf(id_char,"%d",grpid);
      strcat(grp_msg,id_char);
      strcat(grp_msg,"\n");
      n=write(client_socket,grp_msg,sizeof(grp_msg));
      bzero(buffer,256);

      return NULL;

}
//  /removefromgroup <group id> <client id1> <client id2> ... <client idn> : The admin
// should be able to remove member(s) from the group.
```

```
if(buffer[0]=='/'&&buffer[1]=='r'&&buffer[2]=='e'&&buffer[3]=='m'&&buffer[4]=='o'&&buffer[5]=='v'&&buffer[6]=='e
'&&buffer[7]=='f'&&buffer[8]=='r'&&buffer[9]=='o'&&buffer[10]=='m'&&buffer[11]=='g'&&buffer[12]=='r'&&buffer[13]
=='o'&&buffer[14]=='u'&&buffer[15]=='p'&&buffer[16]==' ')
{
        int  grpid;
        int  av_flg=0;



        // for(int i=0;i<100;i++)
        // {
        //      if(group[i].used==0)
        //      {
        //         grpid=i;
        //         group[i].used=1;
        //         av_flg=1;
        //         break;
        //      }
        // }

        // if(av_flg==0)
        // {
        //     char not_msg[256];
        //     strcpy(not_msg,"sorry could not create a group\n");
        //     n=write(client_socket,not_msg,sizeof(not_msg));
        //     bzero(buffer,256);

        //     return NULL;


        //              // send message to requesting client that the group can't be formed
        //              return NULL;
        // }
```

```
// group[grpid].admins[client_socket]=1;
// group[grpid].members[client_socket]=1;


int count=0;
int val=0;
for(int i=16;buffer[i]!='\n';)
{
    if(buffer[i]==' ')
    {
        i+=1;
        val=0;
        while(buffer[i]!=' '&&buffer[i]!='\n')
        {
            val=val*10+(buffer[i]-'0');
            i+=1;
        }
        if(count==0)
        { count+=1;
          grpid=val;

          if(group[grpid].admins[client_socket]==0)
            {
                 char mssg[256];
                strcpy(mssg,"Sorry you are not admin of the group!");
                 n=write(client_socket,mssg,sizeof(mssg));
                 bzero(buffer,256);
                return NULL;
            }
        }
         else
         {
```

```
                if(group[grpid].admins[client_socket]==1)
                {
                  int count=0;
                  for(int t=0;t<100005;i++)
                  {
                      if(group[grpid].admins[t]==1)
                             count+=1;
                  }

                  if(count==1)
                  {
                      group[grpid].used=0;
                      for(int i=0;i<100005;i++)
                      {
                          group[grpid].admins[i]=0;
                          group[grpid].members[i]=0;
                      }
                  }

                 }
               group[grpid].members[val]=0;      // remove members to the groupid;
                }

               if(buffer[i]=='\n')
                   break;
           }
       }

       char id_char[256];
       char grp_msg[256];
       strcpy(grp_msg,"cleints are removed from the group with group id=");
       sprintf(id_char,"%d",grpid);
```

```
        strcat(grp_msg,id_char);
        strcat(grp_msg,"\n");
        n=write(client_socket,grp_msg,sizeof(grp_msg));
        bzero(buffer,256);

        return NULL;

}

// /makegroupbroadcast <group id>: Any admin of the group should be able to modify
// the type of group as broadcast-only in which only admins are allowed to message

if(buffer[0]=='/'&&buffer[1]=='m'&&buffer[2]=='a'&&buffer[3]=='k'&&buffer[4]=='e'&&buffer[5]=='g'&&buffer[6]=='r
'&&buffer[7]=='o'&&buffer[8]=='u'&&buffer[9]=='p'&&buffer[10]=='b'&&buffer[11]=='r'&&buffer[12]=='o'&&buffer[13]
=='a'&&buffer[14]=='d'&&buffer[15]=='c'&&buffer[16]=='a'&&buffer[17]=='s'&&buffer[18]=='t'&&buffer[19]==' ')
{
    int grpid=0;
    int i=20;
    int val=0;
    for(;buffer[i]!=' '&&buffer[i]!='\n';i++)
    {
        val=val*10+(buffer[i]-'0');
    }
    grpid=val;

    if(group[grpid].admins[client_socket]==0)
    {
         char mssg[256];
        strcpy(mssg,"Sorry you are not admin of the group!");
        n=write(client_socket,mssg,sizeof(mssg));
        bzero(buffer,256);
        return NULL;
```

```
    }
    else
    {
    group[grpid].bonly=1;
     char mssg[256];
    strcpy(mssg,"the group is converted to broadcast_only!");
    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;
    }

     bzero(buffer,256);
    return NULL;
}

// . /activegroups : To display all the groups that are currently active on the server and the
// sender is a part of. Here you have to display at the client side the group ids followed by
// the group admin’s client id, and the ids of all the clients that are part of this group

if(buffer[0]=='/'&&buffer[1]=='a'&&buffer[2]=='c'&&buffer[3]=='t'&&buffer[4]=='i'&&buffer[5]=='v'&&buffer[6]=='e
'&&buffer[7]=='g'&&buffer[8]=='r'&&buffer[9]=='o'&&buffer[10]=='u'&&buffer[11]=='p'&&buffer[12]=='s')
{
    int flg=0;
    char mssg[500];
    strcpy(mssg,"Active groups are: ");
    // printf("Active groups are: ");
    for(int i=0;i<100;i++)
    {
          if(group[i].used==1&&group[i].members[client_socket]==1)
          {
             char val_char[256];
```

```
        // sprintf(id_char,"%d",grpid);
        sprintf(val_char,"%d ",i);
        strcat(mssg,val_char);
        strcat(mssg,"(");
        for(int j=0;j<100005;j++)
        {
            if(group[i].admins[j]==1)
            {
                char temp[256];
                sprintf(temp,"%d",j);
                strcat(mssg,temp);
                strcat(mssg,"admin, ");

            }
            else
            if(group[i].members[j]==1)
            {
                 char temp[256];
                sprintf(temp,"%d",j);
                strcat(mssg,temp);
                strcat(mssg,", ");

            }

        }
        strcat(mssg,")\n");

        // for(int j=0;j<100005;j++)
        // {

        // }
```

```
            // char val_char[256];
            // // sprintf(id_char,"%d",grpid);
            // sprintf(val_char,"%d ",i);
            // strcat(mssg,val_char);
          flg=1;
          }
    }
    printf("\n");
    strcat(mssg,"\n");
    if(flg==0)
      {
       strcat(mssg,"NONE\n");
      }

    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;
}

// . /quit : The client will be removed from the server. This client will be removed from all the
// active groups
if(buffer[0]=='/'&&buffer[1]=='q'&&buffer[2]=='u'&&buffer[3]=='i'&&buffer[4]=='t')
{
    available_clients[client_socket]=0;
    char mssg[256];
    strcpy(mssg,"Client is removed from the server successfully\n");
    n=write(client_socket,mssg,sizeof(mssg));
    bzero(buffer,256);
    return NULL;

}
```

```
if(buffer[0]=='b'&&buffer[1]=='r'&&buffer[2]=='o'&&buffer[3]=='a'&&buffer[4]=='d'&&buffer[5]=='c'&&buffer[6]=='a
'&&buffer[7]=='s'&&buffer[8]=='t'&&buffer[9]=='_')
{
        char buffertemp[256];
        strcpy(buffertemp,buffer);
        char to_broad[256];
        char cl_socket_str[256];
        int k=0;
         sprintf(cl_socket_str,"%d",client_socket);
        // for(int i=0;buffertemp[i]!='\n';i++)
        // {
        //    to_broad[k++]=buffertemp[i];
        // }
        strcpy(to_broad,buffertemp);
        strcat(to_broad,"");
        strcat(to_broad,"(broadcast_by(");
        strcat(to_broad,cl_socket_str);
        strcat(to_broad,"))");

        for(int i=0;i<100005;i++)
        {
            if(available_clients[i]&&i!=client_socket)
            {
                n=write(i,to_broad,sizeof(to_broad));
                 bzero(buffer,256);
            }
        }
         bzero(buffer,256);
```

```
            return NULL;
     }
    if(buffer[0]=='s'&&buffer[1]=='h'&&buffer[2]=='o'&&buffer[3]=='w'&&buffer[4]=='_'&&buffer[5]=='c')
    {
       printf("We are inside the show\n");
       char cl_list[10000];
       strcpy(cl_list,"");
       for(int i=0;i<100005;i++)
       {

           if(available_clients[i]&&i!=client_socket)
           //  if(available_clients[i])
           {
                char avlb[5];


               sprintf(avlb,"%d",i);
               strcat(cl_list,avlb);
               strcat(cl_list,"\n");
           }
       }
       // printf("Below is show list:");
       // printf("%s",cl_list);
       // printf("the client socket is:%d",client_socket);
       // printf("above is show list:");
        n=write(client_socket,cl_list,sizeof(cl_list));
        bzero(buffer,256);
        bzero(cl_list,10000);

        return NULL;
    }
```

```
    if(buffer[0]=='c'&&buffer[1]=='o'&&buffer[2]=='n'&&buffer[3]=='n'&&buffer[4]=='e'&&buffer[5]=='c'&&buffer[6]
=='t'&&buffer[7]=='_'&&buffer[8]>='0'&&buffer[8]<='9')
    {
        int val=0;
        for(int i=8;buffer[i]!=' '&&buffer[i]!='\n';i++)
            val=val*10+(buffer[i]-'0');

        if(!available_clients[val])
          {
            char msg[256];
            strcpy(msg,"Sorry the client is presently offline");
             n=write(client_socket,msg,sizeof(msg));
             connect_this[client_socket]=-1;
            return NULL;
          }


        connect_this[client_socket]=val;
        available_clients[client_socket]=1;
         strcpy(res3str,"Connected_successfully");
        n=write(client_socket,res3str,sizeof(res3str));
        bzero(buffer,256);

        return NULL;

    }

    if(connect_this[client_socket]>=-
1&&buffer[0]=='d'&&buffer[1]=='i'&&buffer[2]=='s'&&buffer[3]=='c'&&buffer[4]=='o'&&buffer[5]=='n'&&buffer[6]=='n
'&&buffer[7]=='e'&&buffer[8]=='c'&&buffer[9]=='t'&&buffer[10]=='_'&&buffer[11]=='m')
    {
```

```
        //  printf("%")
        strcpy(res3str,"Disconnected_successfully");
         n=write(client_socket,res3str,sizeof(res3str));
          bzero(buffer,256);
          connect_this[client_socket]=-1;
          available_clients[client_socket]=0;


          return NULL;
    }

    if(connect_this[client_socket]>=0)
    {
        char mfied[256];
        sprintf(mfied,"%d",client_socket);
        strcat(mfied,":");
        strcat(mfied,buffer);
        n=write(connect_this[client_socket],mfied,sizeof(mfied));
        bzero(buffer,256);

        return NULL;
    }




    // strcpy(res3str,"RECIEVED");
    // n=write(client_socket,res3str,sizeof(res3str));
    // bzero(buffer,256);
```

```
// char actualpath[PATH_MAX+1];
// // read(newsockfd,buffer,sizeof(buffer));
// while((bytes_read=read(client_socket,buffer+msgsize,sizeof(buffer)-msgsize-1)))
// {
//     masgsize+=bytes_read;
//     if(msgsize>BUFSIZE-1|| buffer[msgsize-1]=='\n')
//     break;
// }

// check(bytes_read,"recv error");
// buffer[msgsize-1]=0;
// printf("REQUEST:%s\n",buffer);
// fflush(stdout);

// //validity check
// if(realpath(buffer,actualpath)==NULL)
// {
//     print("ERROR(bad path):%s\n",buffer);
//     close(client_socket);
//     return NULL;
// }

// // FILE *fp=fopen(actualpath,"r");
// // if(fp==NULL)
// // {

// // }

// while((bytes_read== fread(buffer,1,BUFSIZE,fp))>0)
// {
//     write(client_socket,buffer,bytes_read);
// }
```

```
// close(client_socket);
// printf("Closing Connection\n");


return NULL;
```

THANK YOU .