14:332:456 Network Centric Programming, Spring 2012

Gregory Paton
Christopher Jelesnianski

Final Project Report: The Instant *Chatter* Application

Motivation

The motivation for the implementation of this project was to use the various concepts learned in Network Centric Programming and apply them to a project that we have not been yet asked to implement. Given the constraints of this project (i.e. it involves networking, and the short amount of time allocated to complete this project), we brainstormed the idea of creating a small instant messenger application. A small application allows it to be portable and thus advantageous to use on mobile platforms. Compared to the commercially available instant messengers, ours would be much thinner and only have the necessary functionality whereas most have a lot of features not commonly used by the consumer which take up memory and drain power due to the services that need to be maintained in case they are utilized.

What sets our application apart is the ability to send private messages to intended users connected to the same server as well as the ability to create a private chat room where only users who know the password may enter the chat room. To help the user be aware of what other users are present in the current chat room and what chat rooms are available to enter, an “ls” command has also been implemented.

Due to the lack of time, our group focused on implementing functionality of our instant messenger application that it can provide over aesthetics as we believed this would be more important to the consumer overall. If more time was given a User Interface would have been developed to overlay the server and client model created in order to create a more appealing dynamic as an actual encapsulated application. In addition, a nice functionality to add would been to introduce multithreading into our application to make it even more responsive and convert it from a sequential application.

Solution

As mentioned earlier, our application has the following main functions available:

1) **Private Messaging:** This allows a user to send a private message to another user using the delimiter “`--pmsg`” followed by the message in their writing window which will only be displayed on the intended recipients display.

2) **Private ChatRooms:** A user in the “Lobby”, the chat room every user is placed when they log in, can create and/or join private chat rooms using the delimiters “`--create`” and/or “`--join`” in their writing window. Note that the person who creates the

chat room may register a password so that the chat room becomes password protected as the only way to join that chat room is if the correct password is given. This gives users a place to discuss sensitive information without the worry of eavesdropping by an unintended party. Note that the password can be shared using means such as utilizing the private messaging option or by some other medium.

3) **“ls” Functions:** This function is a utility function to let the user be aware of his environment such as what chat rooms are currently open and what users are in the same room as the user in case they forget what the name of the chat room or user was.

To list the users currently present, the command “—ls” can be entered into the writing window.

To list the chat rooms currently open on the server, the command “--lsc” can be entered into the writing window.

In order to utilize these functions outlined above it is also necessary to be able navigate through our application in between chat rooms. To handle this we have implemented an easy to understand mechanism described below.

How Does It Work

The mechanics of our instant messenger application are similar to those of the client server model. Our application consists of three parts:

- The Server
- The Client (writing console)
- The Display

The **Server** is the backend of our entire application taking care of managing both users and chat rooms. When a new user connects to the server a new user structure is created for that user. It is assumed that the client writing console connects first and the display for that client is connected second. This explained in detail later on. A user structure contains only necessary information about the user that just joined including their “username”, and their respective client file descriptor and display file descriptor. In addition, a status indicator is also included to tell whether the user structure is populated with an actual user. The beginning of server has the normal socket initialization followed by the select function in order in implement a TCP model.

The **Client** is the writing console portion of the entire client unit while the display is the echo portion of the entire client. Similar to how actual instant messengers work, the Client unit consists of an area where the user writes what they would like to say to other users within the chat room and a display which shows the user what other users have said. The client is also responsible for recognizing special delimiters in order to signal the server an action request (i.e. sending a private message, creating a private room) has been given by a user and to respond appropriately instead of echoing as usual.

The client parses everything that is inputted into the writing console in case a special delimiter was entered. If one was not entered, a normal packet is created with a data OpCode, but then casted in stream in order to be sent into the socket connected with the server. When receiving any message the first thing the server does is decode the OpCode given within the message. From there a variety of paths may follow including: just echoing the data to everyone or running one of the modularized functions depending on the OpCode before continuing to handle requests. If a special delimiter was entered the client would find this as it parses the inputted data. The client would then create a special packet with the associated OpCode and parameters back to the server. The server would acknowledge this op code and proceed to the associated path to fulfill what was requested by the user.

The way the Client and server communicate is that a hybrid protocol was created between TCP sockets and UDP packets. To understand what we mean, please continue reading as we present an overview followed by an example.

Protocol

Our application is based upon the reliability of TCP by using sockets and also

Our application is based upon the reliability of TCP by using sockets and also implements the flexibility of UDP by incorporating OpCodes within each stream that is sent between the Client unit and the Server. OpCodes are the backbone to our application as this the medium in which every client communicates with the server and vice versa. Our Application will use any available port, for our demonstration we will use the port 5555.

Message Formats

General Message Packet:

```
            2 bytes     1 byte       string
            ------------------------------------------------
           | OpCode |  BlockNum  |  Data                    |
            ------------------------------------------------
*Note BlockNum not implemented since TCP is used.
```

Example:

User logs in with correct parameters. //the user is now connected to the “Lobby” chat room.

User types: “--create” into his writing console.
Server responds” “What would you like to name the Chatroom: (Press Enter to cancel this action.)”
User types “my new ChatRoom”
Server responds: What password does this chat room have:”
User type: “abc123”
Server responds: “ChatRoom has been created.”

This brings us to our most important object: the chat room structure, as well as the array that contains all the chat room structures within the server. The chat room structure contains statistics related to itself such as its “name”, the password to be able to enter it, and the list of users currently within this chat room. In order to navigate between different chat rooms we have devised a simple but tedious mechanism due to the lack of time. The way it is implemented currently is that a user must return to the lobby before connecting to another private chat room; this is executed by using the "--lobby" delimiter. If more time was given, this would be one of the first things we change because we realize this mechanism is tedious for the user.

Finally, in addition to handling the lobby chat room, the server is also responsible for updating all the chat rooms chats currently opened with what people have said and where.

Concepts Applied

This class has taught us many ways that network communication is accomplished across the internet, such as communication via sockets, TCP and UDP, as well as the necessary knowledge to also create a secure application resistant against the most common threats.

Our application uses concepts from class such as utilizing and linking individual file descriptors to sockets in order to connect to the server. By using the select function, we have a I/O multiplexing application that will keep track of all the file descriptors of the clients and displays of the clients in conjunction with the appropriate macros such as FD_ISSET in order to manipulate the descriptor sets in our application. Finally, to the best of our ability, we have securely programmed our application to prevent against any buffer overflows or other attacks in order to crash our application.

Code Appendix

```
/*
Network Centric Programming
Final Project
Greg Paton
Chris Jelesnianski
Work was distributed evenly

////Server.c  ////////////////
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <strings.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/select.h>
#include <errno.h>
#include <time.h>
#include <unistd.h>
#include <ifaddrs.h>
#include <netinet/in.h>
```

```
#include <string.h>

//constants
#define USER_NAME_SIZE 64
#define PACKET_DATA_SIZE 512
#define MAX_NUM_USERS 64
#define MAX_NUM_CHATS 16

//structs
struct user        {
        char username[USER_NAME_SIZE];
        int clientfd;
        int displayfd;
        int set;
};

struct chatroom        {
        char name[USER_NAME_SIZE];
        char password[32];
        //struct user *users;
        struct user users[MAX_NUM_USERS];
        int set;
};

struct c_event{
        uint16_t opcode;
        char* chatRoom;
        char stuff[PACKET_DATA_SIZE];
};

struct event        {
        uint16_t opcode;
        char message[512];
};

//globals
static struct chatroom chatrooms[MAX_NUM_CHATS];

//function prototypes
uint16_t get_opcode(char *recvBuf);
int get_user_name(char *recvBuf, int size, char *user_name);
int get_data(char *recvBuf, int size, char *data);
int createChatRoom(int clientfd, int displayfd);
void joinChatRoom(struct user *public_user);
int get_user_chatroom(int clientfd, struct chatroom *chatrooms);
int send_formatted_message(int clientfd, int displayfd, char *message, struct
user
*public_users);
int send_formatted_message_chat(int clientfd, int displayfd, char *recvBuf,
struct
chatroom chatroom);
int send_pmsg(char *recvBuf, int size, int clientfd, struct user
*public_users);
int send_pmsg_chat(char *recvBuf, int size, int clientfd, struct chatroom
*chatrooms);
int notify_user_exit(int clientfd, struct user *public_users);
int notify_user_exit_chat(int clientfd, struct chatroom chatroom);
int notify_user_enter_chat(int clientfd, struct chatroom chatroom);
void leaveChat(int clientfd, struct user *public_users);
int print_address();
int print_help_info(int displayfd);

int main(int argc, char **argv)        {
        if(argc != 2)   {
        fprintf(stderr, "Usage: %s <port number>\n", argv[0]);
        exit(0);
    }

        int listenfd, connfd;
        int port;
        struct sockaddr_in serverAddr, clientAddr;
        socklen_t length;
        char message[4096];
        char recvBuf[1024];
        fd_set socket_set, temp_set;
        fd_set client_set, display_set;
        int max_clientfd = 0;
        int max_displayfd = 0;
        int maxfd = 0;
        int nready;
        int i, j, k, l;
        int tempfd;
        time_t ticks;
        struct user public_users[MAX_NUM_USERS];
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                public_users[i].set = 0;
        }
        uint16_t opcode;
        char user_name[64];
        char data[512];
        struct event error_packet;
        error_packet.opcode = htons(0);
        struct event ack_packet;
        int found;
        int chatroom_index;
        char name[64];

        //initialize chatrooms
        for(i = 0; i < MAX_NUM_CHATS; ++i)        {
                chatrooms[i].set = 0;
                for(j =0; j < MAX_NUM_USERS; ++j)        {
                        chatrooms[i].users[j].set = 0;
                }
        }

        //get port
        port = atoi(argv[1]);

        //set up address structure
        memset((char*)&serverAddr, 0, sizeof(serverAddr));
        serverAddr.sin_family = AF_INET;
        serverAddr.sin_port = htons(port);
        serverAddr.sin_addr.s_addr = htonl(INADDR_ANY);
        //create socket
        if((listenfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)        {
                perror("socket");
                exit(0);
        }

        //bind socket
        if(bind(listenfd, (struct sockaddr *) &serverAddr,
sizeof(serverAddr)) < 0)        {
                perror("bind");
                exit(0);
        }

        //set socket to listen
        if(listen(listenfd, 5) < 0)        {
                perror("listen");
                exit(0);
        }

        length = sizeof(clientAddr);
        FD_ZERO(&client_set);
        FD_SET(listenfd, &client_set);
        max_clientfd = listenfd;

        print_address();

        while(1)        {
                temp_set = client_set;

                if(select(max_clientfd + 1, &temp_set, NULL, NULL, NULL) < 0)
{
            perror("select");
            exit(0);
        }

                //loop through all file descriptors
                for(i = 0; i <= max_clientfd; ++i)        {
                        found = 0;
                        if(FD_ISSET(i, &temp_set)) {
                                //if new connection
                                if(i == listenfd)        {
                                        //accept new connection
                                        if((tempfd = accept(listenfd, (struct
sockaddr *) &clientAddr, &length)) < 0)        {
                                                perror("accept");
                                                continue;
                                        }
                                        if(read(tempfd, recvBuf,
sizeof(recvBuf)) < 0)        {
                                                perror("read");
                                                exit(0);
                                        }
                                        opcode = get_opcode(recvBuf);
                                        get_user_name(recvBuf,
sizeof(recvBuf), user_name);
                                        //if display is connecting
                                        if(opcode == 2)        {
```

```
                                                            for(j = 0; j < MAX_NUM_USERS;
++j)        {

if(!strcmp(public_users[j].username, user_name))        {
                                                                    found = 1;

public_users[j].displayfd = tempfd;

FD_SET(tempfd, &display_set);
                                                                    //send
acknowledgement to display

ack_packet.opcode = htons(6);

if(write(public_users[j].displayfd, (char *)&ack_packet, sizeof(ack_packet))
< 0)        {

perror("write");

exit(0);
                                                                    }
                                                                    //send
acknowledgement to client

if(write(public_users[j].clientfd, (char *)&ack_packet, sizeof(ack_packet))
< 0)        {

perror("write");

exit(0);
                                                                    }
                                                                    //update
maxfd
                                                                    if(tempfd >
max_displayfd)

max_displayfd = tempfd;
                                                                    break;
                                                            }
                                                    }
                                                    if(!found)        {
                                                            close(tempfd);
                                                            continue;
                                                    }
                                            }
                                            //else if client is connecting
                                            else if(opcode == 1)        {
                                                    FD_SET(tempfd, &client_set);
                                                    for(j = 0; j < MAX_NUM_USERS;
++j)        {

if(public_users[j].set == 0)        {

memcpy(public_users[j].username, user_name,
sizeof(public_users[j].username));

public_users[j].clientfd = tempfd;
```

```
public_users[j].set = 1;
                                                                //update
maxfd
                                                                if(tempfd >
max_clientfd)

max_clientfd = tempfd;
                                                                break;
                                                        }
                                                }
                                        }
                                        //else send error
                                        else    {
                                                if(write(i, (char
*)&error_packet, sizeof(error_packet)) < 0)     {
                                                        perror("write");
                                                        exit(0);
                                                }
                                                close(tempfd);
                                        }
                                }
                                //else, handle current connection
                                else    {
                                        //read from client
                                        if(read(i, recvBuf, sizeof(recvBuf))
< 0)    {
                                                perror("read");
                                                exit(0);
                                        }
                                        opcode = get_opcode(recvBuf);
                                        //if data received
                                        if(opcode == 3)         {
                                                //check if user is in
chatroom
                                                if((chatroom_index =
get_user_chatroom(i, chatrooms)) != -1)         {
                                                        //loop through all
users in chatroom
                                                        for(j = 0; j <
MAX_NUM_USERS; ++j)     {
                                                                //if user is
active, display message to them

if(chatrooms[chatroom_index].users[j].set)      {

send_formatted_message_chat(i,
chatrooms[chatroom_index].users[j].displayfd,
recvBuf,chatrooms[chatroom_index]);
                                                                }
                                                        }
                                                }
                                                //else write to all public
clients
                                                else    {
                                                        for(j = 0; j <=
max_displayfd; ++j)     {
```

```
if(FD_ISSET(j, &display_set)) {

//check if user is a public user
                                                                           for(k
= 0; k < MAX_NUM_USERS; ++k)        {

if(public_users[k].displayfd == j)        {

send_formatted_message(i, j, recvBuf, public_users);

}
                                                                           }
                                                                   }
                                                           }
                                                   }
                                           }
                                           //if client is exiting
                                           else if(opcode == 4)        {
                                                   //find user
                                                   for(j = 0; j < MAX_NUM_USERS;
++j)        {
                                                           //check if public
user

if(public_users[j].clientfd == i)        {
                                                                   //remove user
from sets and public_users

notify_user_exit(i, public_users);

write(public_users[j].displayfd, "--exit", 7);

close(public_users[j].clientfd);

close(public_users[j].displayfd);

FD_CLR(public_users[j].clientfd, &client_set);

FD_CLR(public_users[j].displayfd, &display_set);
                                                                   bzero((char
*)&public_users[j], sizeof(public_users[j]));

                                                                   break;
                                                           }
                                                           for(k = 0; k <
MAX_NUM_CHATS; ++k)        {

if(chatrooms[k].users[j].clientfd == i)        {

//remove user from sets and public_users

notify_user_exit_chat(i, chatrooms[k]);

write(chatrooms[k].users[j].displayfd, "--exit", 7);

close(chatrooms[k].users[j].clientfd);
close(chatrooms[k].users[j].displayfd);

FD_CLR(chatrooms[k].users[j].clientfd, &client_set);

FD_CLR(chatrooms[k].users[j].displayfd, &display_set);

bzero((char *)&chatrooms[k].users[j], sizeof(chatrooms[k].users[j]));

break;
                                                                }
                                                        }
                                                }
                                        }
                                        //create chatroom
                                        else if(opcode == 7){
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {

if(public_users[j].clientfd == i)        {

createChatRoom(public_users[j].clientfd, public_users[j].displayfd);
                                                                break;
                                                        }
                                                }
                                        }
                                        //joinchatroom
                                        else if(opcode == 8){
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {

if(public_users[j].clientfd == i)        {

joinChatRoom(&public_users[j]);
                                                                break;
                                                        }
                                                }
                                        }
                                        //if pmsg command called
                                        else if(opcode == 9)        {
                                                //get users displayfd
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {
                                                        //check public users

if((public_users[j].clientfd == i) && (public_users[j].set != 0))        {
                                                                //send
private message

if(send_pmsg(recvBuf, sizeof(recvBuf), i, public_users) < 0)        {

if(write(public_users[j].displayfd, "--pmsg failed\n", sizeof("--pmsg
failed\n")) < 0)        {

perror("write");

break;
```

```
                                                                        }
                                                                }
                                                                found = 1;
                                                        }
                                                        //check private chat
users
                                                        for(k = 0; k <
MAX_NUM_CHATS; ++k)        {

if((chatrooms[k].users[j].clientfd == i) && (chatrooms[k].users[j].set !=0))
{

if(send_pmsg_chat(recvBuf, sizeof(recvBuf), i, chatrooms) < 0)        {

if(write(chatrooms[k].users[j].displayfd, "--pmsg failed\n",
sizeof("--pmsg failed\n")) < 0)        {

perror("write");

break;

}
                                                                        }
                                                                        found
= 1;

break;
                                                                }
                                                        }
                                                        if(found)
                                                                break;
                                                }
                                        }
                                        //if ls command called
                                        else if(opcode == 10)        {
                                                //get user displayfd
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {
                                                        //check if user is
public

if(public_users[j].clientfd == i)        {
                                                                //loop
through all users and print usernames
                                                                for(k = 0; k
< MAX_NUM_USERS; ++k)        {

if(public_users[k].set)        {

bzero(&message, sizeof(message));

snprintf(message, sizeof(message), "%s\n", public_users[k].username);

if(write(public_users[j].displayfd, message, sizeof(message)) < 0)        {

perror("write");
continue;

}
                                                                        }
                                                                }
                                                                break;
                                                        }
                                                        //check if user is
private
                                                        for(k = 0; k <
MAX_NUM_CHATS; ++k)        {

if(chatrooms[k].users[j].clientfd == i)        {

//loop through all users and print usernames
                                                                        for(l
= 0; l < MAX_NUM_USERS; ++l)        {

if(chatrooms[k].users[l].set)        {

bzero(&message, sizeof(message));

snprintf(message, sizeof(message), "%s\n", chatrooms[k].users[l].username);

if(write(chatrooms[k].users[j].displayfd, message, sizeof(message)) < 0)
{

perror("write");

continue;

}

}
                                                                        }

break;
                                                                }
                                                        }
                                                }
                                        }
                                        //if lsc command called
                                        else if(opcode == 11)        {
                                                //get user displayfd
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {

if(public_users[j].clientfd == i)        {
                                                                //find active
chatrooms
                                                                for(k = 0; k
< MAX_NUM_CHATS; ++k)        {
                                                                        //if
chatroom is active, write to user displayfd

if(chatrooms[k].set != 0)        {
```

```
bzero(&message, sizeof(message));

snprintf(message, sizeof(message), "%s", chatrooms[k].name);

if(write(public_users[j].displayfd, message, sizeof(message)) < 0)        {

perror("write");

continue;

}
                                                                        }
                                                                }
                                                                break;
                                                        }
                                                }
                                        }
                                        //if lobby command called
                                        else if(opcode == 12)        {
                                                leaveChat(i, public_users);
                                        }
                                        //if help command called
                                        else if(opcode == 20)        {
                                                //loop through all users
                                                for(j = 0; j < MAX_NUM_USERS;
++j)        {
                                                        //check public users

if(public_users[j].clientfd == i)        {

print_help_info(public_users[j].displayfd);
                                                                found = 1;
                                                        }
                                                        //check private chat
users
                                                        for(k = 0; k <
MAX_NUM_CHATS; ++k)        {

if(chatrooms[k].users[j].clientfd == i)        {

print_help_info(chatrooms[k].users[j].displayfd);
                                                                        found
= 1;

break;
                                                                }
                                                        }
                                                        if(found)
                                                                break;
                                                }
                                        }
                                }
                        }
                }
        }
        close(connfd);
        close(listenfd);

        return 0;
}

uint16_t get_opcode(char *recvBuf)        {
        int i;
        uint16_t op = 0;
    uint16_t byte1;
        uint16_t byte0;
        char opcode[2];

        //parse the packet
    for(i = 0; i < 2; ++i)  {
        if(i < 2)   {
            opcode[i] = recvBuf[i];
        }
    }

        //convert opcode from char array to uint16_t
    byte0 = opcode[1];
        byte1 = opcode[0];
    byte1 = byte1 << 8;
        op = byte0 | byte1;

        //check opcode is valid
        return op;
}

int get_user_name(char *recvBuf, int size, char *user_name)        {
        int i = 0;
        char curr;
        do       {
                curr = recvBuf[i+2];
                user_name[i] = curr;
                ++i;
        } while(curr != 0);

        return 1;
}

int get_data(char *recvBuf, int size, char *data)        {
        int i = 0;
        char curr;
        do       {
                curr = recvBuf[i+4];
                data[i] = curr;
                ++i;
        } while(curr != 0);

        return 1;
}

int createChatRoom(int clientfd, int displayfd){
        char ask[] = "Create chatroom name:  (\"--cancel\" to cancel)\n";
        char ask2[] = "Create chatroom password:\n";
        char ask3[] = "Re-enter password:\n";
        char err[] = "Name already in use. Choose new name: (\"--cancel\" to
cancel) \n";
        char err2[] = "Passwords do not match! Re-enter password:\n";
        char ack[]= "ChatRoom has been created.\n";
        char cancel[] = "Action cancelled\n";
        char name[64];
        char password[64];
        char temp[64];
        int len;
        int n;
        int nread;
        int open_index;
        int same = 1;

        //find available chatroom in array
        for(n = 0; n < MAX_NUM_CHATS; ++n)        {
                if(!chatrooms[n].set)        {
                        open_index = n;
                        break;
                }
        }
        //ask user for chatroom name
        if(write(displayfd, ask, sizeof(ask)) < 0){
                perror("write");
                exit(0);
        }

        while(same)        {
                same = 0;
                //read name from client
                bzero(&name, sizeof(name));
                if(read(clientfd, name, sizeof(name)) < 0)        {
                        perror("read");
                        return -1;
                }
                //if action canceled by user
                if(!strcmp(name+4, "--cancel\n")){
                        if(write(displayfd, cancel, sizeof(cancel)) < 0){
                                perror("write");
                                exit(0);
                        }
                        return 0;
                }
                //check if name is available
                for(n = 0; n < MAX_NUM_CHATS; ++n){
                        if((!strcmp(name+4, chatrooms[n].name)) &&
(chatrooms[n].set != 0)){
                                same = 1;
                                if(write(displayfd, err, sizeof(err)) < 0){
                                        perror("write");
                                        return -1;
                                }
                                break;
                        }
                }
        }
        //set chatroom name
        memcpy(chatrooms[open_index].name, name+4,
sizeof(chatrooms[open_index].name));
        //ask user to set password
        if(write(displayfd, ask2, sizeof(ask2)) < 0){
                perror("write");
                exit(0);
        }
        while(1)        {
                //read password from client
                bzero(&temp, sizeof(temp));
                while(get_opcode(temp) != 3)        {
                        if(read(clientfd, temp, sizeof(temp)) < 0)        {
                                perror("read");
                                exit(0);
                        }
                }
                //if action canceled by user
                if(!strcmp(temp+4, "--cancel\n")){
                        if(write(displayfd, cancel, sizeof(cancel)) < 0){
                                perror("write");
                                exit(0);
                        }
                        return 0;
                }
                memcpy(password, temp+4, sizeof(password));
                //ask user to re-enter password
                if(write(displayfd, ask3, sizeof(ask3)) < 0){
                        perror("write");
                        exit(0);
                }
                //read password again
                bzero(&temp, sizeof(temp));
                while(get_opcode(temp) != 3)        {
                        if(read(clientfd, temp, sizeof(temp)) < 0)        {
                                perror("read");
                                exit(0);
                        }
                }
                //if action canceled by user
                if(!strcmp(temp+4, "--cancel\n")){
                        if(write(displayfd, cancel, sizeof(cancel)) < 0){
                                perror("write");
                                exit(0);
                        }
                        return 0;
                }
                if(!strcmp(temp+4, password))        {
                        break;
                }
                if(write(displayfd, err2, sizeof(err2)) < 0){
                        perror("write");
                        exit(0);
                }
        }
        //set chatroom password
        memcpy(chatrooms[open_index].password, password,
sizeof(chatrooms[open_index].password));
        //set chatroom as active
        chatrooms[open_index].set = 1;
        //send confirmation
        if(write(displayfd, ack, sizeof(ack)) < 0){
                perror("write");
                exit(0);
        }
        return 0;
} //end createChatRoom

void joinChatRoom(struct user *public_user){
        char ask[]= "What Chat Room would you like to join: (\"--cancel\" to
cancel)\n";
        char ack1[128];
        char ack2[]= "Joining chatroom failed.\n";
        char ack3[]= "Password incorrect. Returning to public chatroom.\n";
        char askpass[]= "Password: \n";
        char name[64];
        char password[64];
        bzero(&password, sizeof(password));
        int i, j, k;
        int attempts = 0;
        int open_index;
        int chatroom_index;


        //ask user which chatroom to join
        if(write(public_user->displayfd, ask, sizeof(ask)) < 0){
                perror("write");
                exit(0);
        }
        //get chatroom name
        if(read(public_user->clientfd, name, sizeof(name)) < 0)        {
                perror("read");
                exit(0);
        }
        //check if name exists
        for(i = 0; i < MAX_NUM_CHATS; ++i){
                if((!strcmp(name+4, chatrooms[i].name)) && (chatrooms[i].set
!= 0)){
                        chatroom_index = i;
                        //find available user in chatroom array
                        for(j = 0; j < MAX_NUM_CHATS; ++j)        {
                                if(!chatrooms[i].users[j].set)        {
                                        open_index = j;
                                        break;
                                }
                        }
                        //ask user for password
                        if(write(public_user->displayfd, askpass,
sizeof(askpass)) < 0){
                                perror("write");
                                exit(0);
                        }
                        //get chatroom password
                        while(get_opcode(password) != 3)        {
                                if(read(public_user->clientfd, password,
sizeof(password)) < 0)        {
                                        perror("read");
                                        exit(0);
                                }
                        }
                        //if action canceled by user, return
                        if(!strcmp(password, "\n")){
                                printf("Join Action Canceled\n");
                                return;
                        }
                        //if password not the same, reject
                        if(strcmp(password+4, chatrooms[i].password)){
                                if(write(public_user->displayfd , ack3,
sizeof(ack3)) < 0){
                                        perror("write");
                                        exit(0);
                                }
                                return;
                        }
                        //add user to chat room
                        memcpy(chatrooms[i].users[open_index].username,
public_user->username,
sizeof(chatrooms[i].users[open_index].username));
                        chatrooms[i].users[open_index].set = 1;
                        chatrooms[i].users[open_index].clientfd =
public_user->clientfd;
                        chatrooms[i].users[open_index].displayfd =
public_user->displayfd;
                        // notify user of success
                        snprintf(ack1, sizeof(ack1), "User moved to %s",
name+4);
                        if(write(public_user->displayfd, ack1, sizeof(ack1))
< 0){
                                perror("write");
                                return;
                        }
                        //notify other users
                        notify_user_enter_chat(public_user->clientfd,
chatrooms[chatroom_index]);
                        //remove user from public users
                        bzero(public_user, sizeof(struct user));

                        return;
                }
        }

        // notify failure
        if(write(public_user->displayfd, ack2, sizeof(ack2)) < 0){
                perror("write");
                exit(0);
        }
} //end joinChatRoom

//finds which chatroom the user corresponding to clientfd is located
//returns -1 if user not found in chatroom
int get_user_chatroom(int clientfd, struct chatroom *chatrooms)        {
        int index = -1;
        int i, j;

        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                for(j = 0; j < MAX_NUM_CHATS; ++j)        {
                        if((chatrooms[j].users[i].clientfd == clientfd) &&
(chatrooms[j].set != 0))        {
                                index = j;
                                break;
                        }
                }
        }

        return index;
}

int send_formatted_message(int clientfd, int displayfd, char *recvBuf, struct
user
*public_users)        {
        char message[4096];
        char data[512];
        time_t ticks;
        int k;

        //get system time
        if((ticks = time(NULL)) < 0)        {
                perror("time");
                exit(0);
        }
        bzero(&message, sizeof(message));
        get_data(recvBuf, sizeof(recvBuf), data);
        //get user name
        for(k = 0; k < MAX_NUM_USERS; ++k)        {
                if(public_users[k].clientfd == clientfd)        {
                        snprintf(message, sizeof(message), "[%.8s]%s: %s",
ctime(&ticks)+11,
public_users[k].username, data);
                        break;
                }
        }
        //send message
        if(write(displayfd, message, sizeof(message)) < 0)        {
                perror("write");
                exit(0);
        }
}

int send_formatted_message_chat(int clientfd, int displayfd, char *recvBuf,
struct
chatroom chatroom)        {
        char message[4096];
        char data[512];
        time_t ticks;
        int k;

        //get system time
        if((ticks = time(NULL)) < 0)        {
                perror("time");
                exit(0);
        }
        bzero(&message, sizeof(message));
        get_data(recvBuf, sizeof(recvBuf), data);
        //get user name
        for(k = 0; k < MAX_NUM_USERS; ++k)        {
                if(chatroom.users[k].clientfd == clientfd)        {
                        //format message
                        snprintf(message, sizeof(message), "[%.8s]%s: %s",
ctime(&ticks)+11,
chatroom.users[k].username, data);
                        break;
                }
        }
        //send message
        if(write(displayfd, message, sizeof(message)) < 0)        {
                perror("write");
                exit(0);
        }
}

int send_pmsg(char *recvBuf, int size, int clientfd, struct user
*public_users)        {
        char message[4096];
        char data[512];
        char name[64];
        char curr;
        time_t ticks;
        int found = -1;
        int i, j, k;
        int spaces_found = 0;
        //get system time
        if((ticks = time(NULL)) < 0)        {
                perror("time");
                exit(0);
        }
        bzero(&message, sizeof(message));
        bzero(&name, sizeof(name));
        bzero(&data, sizeof(data));
        //get user name and message
        j = 0;
        k = 0;
        for(i = 4; i < size; ++i)        {
                curr = recvBuf[i];
                if((curr == ' ') && (spaces_found < 2))
                        ++spaces_found;
                else if(curr == 0)
                        break;
                else if(spaces_found == 1)        {
                        name[j] = curr;
                        ++j;
                }
                else if(spaces_found > 1)        {
                        data[k] = curr;
                        ++k;
                }
        }
        //get user name
        for(k = 0; k < MAX_NUM_USERS; ++k)        {
                if(public_users[k].clientfd == clientfd)        {
                        //format message
                        snprintf(message, sizeof(message), "[%.8s]private
message from %s: %s",
ctime(&ticks)+11, public_users[k].username, data);
                        break;
                }
        }
        //check if user is a public user
        for(k = 0; k < MAX_NUM_USERS; ++k)        {
                if(!strcmp(public_users[k].username, name) &&
(public_users[k].set != 0))        {
                        if(write(public_users[k].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                                return -1;
                        }
                        found = 0;
                        break;
                }
        }
        //if nothing found, return error
        if(found == -1)
                return found;
        for(j = 0; j < MAX_NUM_USERS; ++j)        {
                if(public_users[j].clientfd == clientfd)        {
                        //format and display message to sender
                        bzero(&message, sizeof(message));
                        snprintf(message, sizeof(message), "[%.8s]pmsg from
%s to %s: %s",
ctime(&ticks)+11, public_users[j].username, public_users[k].username, data);
                        if(write(public_users[j].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                                return -1;
                        }
                        break;
                }
        }
        return found;
}

int send_pmsg_chat(char *recvBuf, int size, int clientfd, struct chatroom
*chatrooms)        {
        char message[4096];
        char data[512];
        char name[64];
        char curr;
        time_t ticks;
        int found = -1;
        int i, j, k;
        int spaces_found = 0;
        int chatroom_index;
        //get system time
        if((ticks = time(NULL)) < 0)        {
                perror("time");
                exit(0);
        }
        bzero(&message, sizeof(message));
        bzero(&name, sizeof(name));
        bzero(&data, sizeof(data));
        //get user name and message
        j = 0;
        k = 0;
        for(i = 4; i < size; ++i)        {
                curr = recvBuf[i];
                if((curr == ' ') && (spaces_found < 2))
                        ++spaces_found;
                else if(curr == 0)
                        break;
                else if(spaces_found == 1)        {
                        name[j] = curr;
                        ++j;
                }
                else if(spaces_found > 1)        {
                        data[k] = curr;
                        ++k;
                }
        }
        //get user name
        for(i = 0; i < MAX_NUM_CHATS; ++i)        {
                for(j = 0; j < MAX_NUM_USERS; ++j)        {
                        if(chatrooms[i].users[j].clientfd == clientfd)
{
                                chatroom_index = i;
                                //format message
                                snprintf(message, sizeof(message),
"[%.8s]private message from %s: %s",
ctime(&ticks)+11, chatrooms[i].users[j].username, data);
                                break;
                        }
                }
        }
        //check if user is a private chat user
        for(k = 0; k < MAX_NUM_USERS; ++k)        {
                if(!strcmp(chatrooms[chatroom_index].users[k].username, name)
&&
(chatrooms[chatroom_index].users[k].set != 0))        {

if(write(chatrooms[chatroom_index].users[k].displayfd, message,
sizeof(message))
< 0)        {
                                perror("write");
                                return -1;
                        }
                        found = 0;
                        break;
                }
        }
        //if nothing found, return error
        if(found == -1)
```

```
                return found;
        for(j = 0; j < MAX_NUM_USERS; ++j)        {
                if(chatrooms[chatroom_index].users[j].clientfd == clientfd)
{
                        //format and display message to sender
                        bzero(&message, sizeof(message));
                        snprintf(message, sizeof(message), "[%.8s]pmsg from
%s to %s: %s",
ctime(&ticks)+11, chatrooms[chatroom_index].users[j].username,
chatrooms[chatroom_index].users[k].username, data);

if(write(chatrooms[chatroom_index].users[j].displayfd, message,
sizeof(message))
< 0)        {
                                perror("write");
                                return -1;
                        }
                        break;
                }
        }
        return found;
}

int notify_user_exit(int clientfd, struct user *public_users)        {
        int i;
        int found = -1;
        char message[128];

        //get username
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if(public_users[i].clientfd == clientfd)        {
                        //format message
                        snprintf(message, sizeof(message), "%s has exited\n",
public_users[i].username);
                        found = 0;
                        break;
                }
        }
        //if not found, return error
        if(found == -1)
                return found;
        //display exit notification to all other users
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if((public_users[i].clientfd != clientfd) &&
(public_users[i].set !=0))        {
                        if(write(public_users[i].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                        }
                }
        }

        return found;
}

int notify_user_exit_chat(int clientfd, struct chatroom chatroom)        {
        int i;
        int found = -1;
        char message[128];

        //get username
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if(chatroom.users[i].clientfd == clientfd)        {
                        //format message
                        snprintf(message, sizeof(message), "%s has exited\n",
chatroom.users[i].username);
                        found = 0;
                        break;
                }
        }
        //if not found, return error
        if(found == -1)
                return found;
        //display exit notification to all other users
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if((chatroom.users[i].clientfd != clientfd) &&
(chatroom.users[i].set !=0))        {
                        if(write(chatroom.users[i].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                        }
                }
        }

        return found;
}

int notify_user_enter_chat(int clientfd, struct chatroom chatroom)        {
        int i;
        int found = -1;
        char message[128];

        //get username
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if(chatroom.users[i].clientfd == clientfd)        {
                        //format message
                        snprintf(message, sizeof(message), "%s has
entered\n", chatroom.users[i].username);
                        found = 0;
                        break;
                }
        }
        //if not found, return error
        if(found == -1)
                return found;
        //display exit notification to all other users
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if((chatroom.users[i].clientfd != clientfd) &&
(chatroom.users[i].set !=0))        {
                        if(write(chatroom.users[i].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                        }
                }
        }

        return found;
}

void leaveChat(int clientfd, struct user *public_users){
        int q=-1;
        int w=-1;
        int e=-1;
        int chatroomppl;
        int i;
        int j;
        char message[256];

        for(i = 0; i < MAX_NUM_CHATS; ++i){
                chatroomppl=0;
                for(j =0; j < MAX_NUM_USERS; ++j){
                        chatroomppl+=chatrooms[i].users[j].set;
                        if(chatrooms[i].set!=0 &&
chatrooms[i].users[j].set!=0 &&
chatrooms[i].users[j].clientfd ==clientfd)
                        {
                                q=i; //which chat room they are in
                                w=j; // which user index they are in withing
that chat room
                        }
                }
                if(q!=-1)
                        break;
        }
        for(j =0; j < MAX_NUM_USERS; ++j){
                if(public_users[j].set==0)
                {
                        e=j; //first empty stop within public_users.
                        break;
                }
        }

        if(q==-1){ //user not found.
                exit(0);

        }
        memcpy((char *)&public_users[e], (char
*)&chatrooms[q].users[w],sizeof(struct user));
        //chatrooms[q].users[w].set=0;
        //remove user from chat
        bzero((char *)&chatrooms[q].users[w], sizeof(chatrooms[q].users[w]));

        chatroomppl--;
        //if user is last user within this chatroom, delete the chatroom.
update chatroom
array
        if(!chatroomppl)
        {
                chatrooms[q].set=0;
        }
        //notify other users
        bzero(&message, sizeof(message));
        snprintf(message, sizeof(message), "%s has left the chat\n",
public_users[e].username);
        for(i = 0; i < MAX_NUM_USERS; ++i)        {
                if(chatrooms[q].users[i].set)        {
                        if(write(chatrooms[q].users[i].displayfd, message,
sizeof(message)) < 0)        {
                                perror("write");
                                return;
                        }
                }
        }
        //notify user
        if(write(public_users[e].displayfd, "now in lobby\n", sizeof("now in
lobby\n")) < 0)        {
                perror("write");
                return;
        }
}

int print_address()        {
        struct ifaddrs * ifAddrStruct=NULL;
    struct ifaddrs * ifa=NULL;
    void * tmpAddrPtr=NULL;

    getifaddrs(&ifAddrStruct);

    for(ifa = ifAddrStruct; ifa != NULL; ifa = ifa->ifa_next) {
        if(ifa->ifa_addr->sa_family==AF_INET) { // check it is IP4
            // is a valid IP4 Address
            tmpAddrPtr=&((struct sockaddr_in *)ifa->ifa_addr)->sin_addr;
            char addressBuffer[INET_ADDRSTRLEN];
            inet_ntop(AF_INET, tmpAddrPtr, addressBuffer, INET_ADDRSTRLEN);
                        if(!strcmp(ifa->ifa_name, "lo"))
                                continue;
            printf("%s IP: %s\n", ifa->ifa_name, addressBuffer);
                }
    }
    if(ifAddrStruct!=NULL)
                freeifaddrs(ifAddrStruct);
    return 0;
}

int print_help_info(int displayfd)        {
        char message[4096] = "Commands:\n\t--create\t\tcreate a private chat
room\n\t--pmsg
<user name>\t\tsend a private message to a user\n\t--exit\t\texit the public
chatroom\n\t--join\t\tjoin an existing chatroom\n\t--ls\t\tlist
users\n\t--lsc\t\tlist chatrooms\n\t--lobby\t\tmove from privat chat to
lobby\n\t--help\t\tdisplay this information\n";

        if(write(displayfd, message, sizeof(message)) < 0)        {
                perror("write");
                return -1;
        }
        return 0;
}
```

Client.c

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

#define USER_NAME_SIZE 32
#define PACKET_DATA_SIZE 512

struct c_connect_client        {
        uint16_t opcode;
        char user_name[USER_NAME_SIZE];
};

struct c_data        {
        uint16_t opcode;
        uint16_t blocknum;
        char data[PACKET_DATA_SIZE];
};

struct c_event{
        uint16_t opcode;
        char* chatRoom;
        char stuff[PACKET_DATA_SIZE];
};

int ParseAndCreateEvent(char *input, struct c_event *eventHandle);
uint16_t get_opcode(char *recvBuf);


int main(int argc, char **argv)        {
        if(argc > 1)        {
                if(!strcmp(argv[1], "--help"))        {
                        printf("Usage: %s <user name> <address> <port number>
[options]\n", argv[0]);
                        printf("Options:\n");
                        printf("\t--help\t\tdisplay this information\n");
                        printf("Commands:\n");
                        printf("\t--create\t\tcreate a private chat room\n");
                        printf("\t--pmsg <user name>\t\tsend a private
message to a user\n");
                        printf("\t--exit\t\texit the public chatroom\n");
                        printf("\t--join\t\tjoin an existing chatroom\n");
                        printf("\t--ls\t\tlist users\n");
                        printf("\t--lsc\t\tlist chatrooms\n");
                        printf("\t--lobby\t\tmove from privat chat to
lobby\n");
                        printf("\t--help\t\tdisplay commands\n");
                        exit(0);
                }
        }

        if(argc < 4)    {
        fprintf(stderr, "Usage: %s <user name> <address> <port number>
[options]\n",
argv[0]);
        exit(0);
    }

        int sockfd;
        int port;
        struct sockaddr_in addr;
        char address[24];
        char input[1024];
        char message[1024];
        char user_name[64];
        time_t ticks;
        int i;
        int opcode;
        struct c_data data_packet;
        struct c_event event_handle;
        char recvBuf[516];

        //get port
        port = atoi(argv[3]);
        strncpy(user_name, argv[1], sizeof(user_name));

        //set up address structure
        memset((char*)&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons(port);
        inet_pton(AF_INET, argv[2], &addr.sin_addr);

        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)        {
                perror("socket");
                exit(0);
        }

        if(connect(sockfd, (struct sockaddr *) &addr, sizeof(addr)) < 0)
{
                perror("connect");
                exit(0);
        }

        //send user name to server to request to join
        struct c_connect_client packet;
        packet.opcode = htons(1);
        memcpy(packet.user_name, user_name, sizeof(packet.user_name));
        if(write(sockfd, (char *)&packet, sizeof(packet)) < 0)        {
                perror("write");
                exit(0);
        }
        printf("waiting for display connection...\n");
        if(read(sockfd, recvBuf, sizeof(recvBuf)) < 0)        {
                perror("read");
                exit(0);
        }
        if(get_opcode(recvBuf) != 6)        {
                fprintf(stderr, "could not connect\n");
                exit(0);
        }

        while(1)        {
                //clear input display
                for(i = 0; i < 50; ++i)        {
                        printf("\n");
                }
                //get user input
                fgets(input, sizeof(input), stdin);

                //if user entered input
                if(input[0] != '\n')        {
                        //ParseAndCreateEvent(input, &event_handle);
                        if(!strcmp(input, "--exit\n"))
                                opcode = 4;
                        else if(!strcmp(input, "--create\n"))
                                opcode = 7;
                        else if(!strcmp(input, "--join\n"))
                                opcode = 8;
                        else if(strstr(input, "--pmsg") != NULL)
                                opcode = 9;
                        else if(!strcmp(input, "--ls\n"))
                                opcode = 10;
                        else if(!strcmp(input, "--lsc\n"))
                                opcode = 11;
                        else if(!strcmp(input, "--lobby\n"))
                                opcode = 12;
                        else if(!strcmp(input, "--help\n"))
                                opcode = 20;
                        else
                                opcode = 3;


                        //create packet
                        bzero(&data_packet, sizeof(data_packet));
                        data_packet.opcode = htons(opcode);
                        data_packet.blocknum = htons(0);
                        memcpy(data_packet.data, input, 
sizeof(data_packet.data));
                        //send message
                        if(write(sockfd, (char *)&data_packet, 
sizeof(data_packet)) < 0)        {
                                perror("write");
                                exit(0);
                        }

                        //bzero(&event_handle, sizeof(event_handle));

                        //check for input commands
                        if(!strcmp(input, "--exit\n"))        {
                                break;
                        }
                }
        }

        close(sockfd);

        return 0;
}

uint16_t get_opcode(char *recvBuf)        {
        int i;
        uint16_t op = 0;
    uint16_t byte1;
        uint16_t byte0;
        char opcode[2];

        //parse the packet
    for(i = 0; i < 2; ++i)  {
        if(i < 2)   {
            opcode[i] = recvBuf[i];
        }
    }

        //convert opcode from char array to uint16_t
    byte0 = opcode[1];
        byte1 = opcode[0];
    byte1 = byte1 << 8;
        op = byte0 | byte1;

        //check opcode is valid
        return op;
}

//parses and creates packet depending on user input.
int ParseAndCreateEvent(char *input, struct c_event *eventHandle){

        char* tempPtr;
        tempPtr = strstr(input, "--create");
        if(tempPtr != NULL ){
        //join command found create join packet
                eventHandle->opcode = htons(7);
                printf("op: %d___", eventHandle->opcode);

                eventHandle->chatRoom = tempPtr+9;
                printf("name: %s___", eventHandle->chatRoom);
                return 1;
        }
        tempPtr = strstr(input, "--join");
        if(tempPtr != NULL ){
        //join command found create join packet
                eventHandle->opcode = htons(8);
                printf("op: %d___", eventHandle->opcode);

                eventHandle->chatRoom = tempPtr+7;
                printf("name: %s___", eventHandle->chatRoom);
                return 1;
        }
        tempPtr = strstr(input, "--pmsg");
        if(tempPtr != NULL ){
        //join command found create join packet
                eventHandle->opcode = htons(9);
                printf("op: %d___", eventHandle->opcode);

                eventHandle->chatRoom = tempPtr+7;
                printf("name: %s___", eventHandle->chatRoom);
                return 1;
        }

        char temp;
        int i=0;
        return 0;
}
```

Display.c

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

#define USER_NAME_SIZE 32
#define PACKET_DATA_SIZE 512

struct c_connect_client        {
        uint16_t opcode;
        char user_name[USER_NAME_SIZE];
};

struct c_data        {
        uint16_t opcode;
        uint16_t blocknum;
        char data[PACKET_DATA_SIZE];
};

uint16_t get_opcode(char *recvBuf);

int main(int argc, char **argv)        {

        if(argc != 4)   {
        fprintf(stderr, "Usage: %s <user name> <address> <port number>\n",
argv[0]);
        exit(0);
    }

        int sockfd;
        int port;
        struct sockaddr_in addr;
        char recvBuf[1024];
        fd_set socket_set;
        int maxfd;
        char user_name[64];
        int i;

        //get port
        port = atoi(argv[3]);
        strncpy(user_name, argv[1], sizeof(user_name));

        //set up address structure
        memset((char*)&addr, 0, sizeof(addr));
        addr.sin_family = AF_INET;
        addr.sin_port = htons(port);
        //addr.sin_addr.s_addr = htonl(INADDR_ANY);
        inet_pton(AF_INET, argv[2], &addr.sin_addr);
        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)        {
                perror("socket");
                exit(0);
        }

        if(connect(sockfd, (struct sockaddr *) &addr, sizeof(addr)) < 0)
{
                perror("connect");
                exit(0);
        }

        struct c_connect_client packet;
        packet.opcode = htons(2);
        memcpy(packet.user_name, user_name, sizeof(packet.user_name));
        if(write(sockfd, (char *)&packet, sizeof(packet)) < 0)        {
                perror("write");
                exit(0);
        }

        if(read(sockfd, recvBuf, sizeof(recvBuf)) < 0)        {
                perror("read");
                exit(0);
        }
        if(get_opcode(recvBuf) != 6)        {
                fprintf(stderr, "could not connect\n");
                close(sockfd);
                exit(0);
        }

        FD_ZERO(&socket_set);
        FD_SET(sockfd, &socket_set);
        maxfd = sockfd;

        //clear output display
        for(i = 0; i < 50; ++i)        {
                printf("\n");
        }

        while(1)        {
                if(select(maxfd + 1, NULL, &socket_set, NULL, NULL) < 0) {
            perror("select");
            exit(0);
        }

                if(FD_ISSET(sockfd, &socket_set)) {
                        if(read(sockfd, recvBuf, sizeof(recvBuf)) < 0)
{
                                perror("write");
                                exit(0);
                        }
                }
                if(!strcmp(recvBuf, "--exit"))        {
                        break;
                }
                printf("%s", recvBuf);
                fflush(stdout);
        }

        close(sockfd);

        return 0;
}

uint16_t get_opcode(char *recvBuf)        {
        int i;
        uint16_t op = 0;
    uint16_t byte1;
        uint16_t byte0;
        char opcode[2];

        //parse the packet
    for(i = 0; i < 2; ++i)  {
        if(i < 2)   {
            opcode[i] = recvBuf[i];
        }
    }

        //convert opcode from char array to uint16_t
    byte0 = opcode[1];
        byte1 = opcode[0];
    byte1 = byte1 << 8;
        op = byte0 | byte1;

        //check opcode is valid
        return op;
}
```