Machine Problem 2

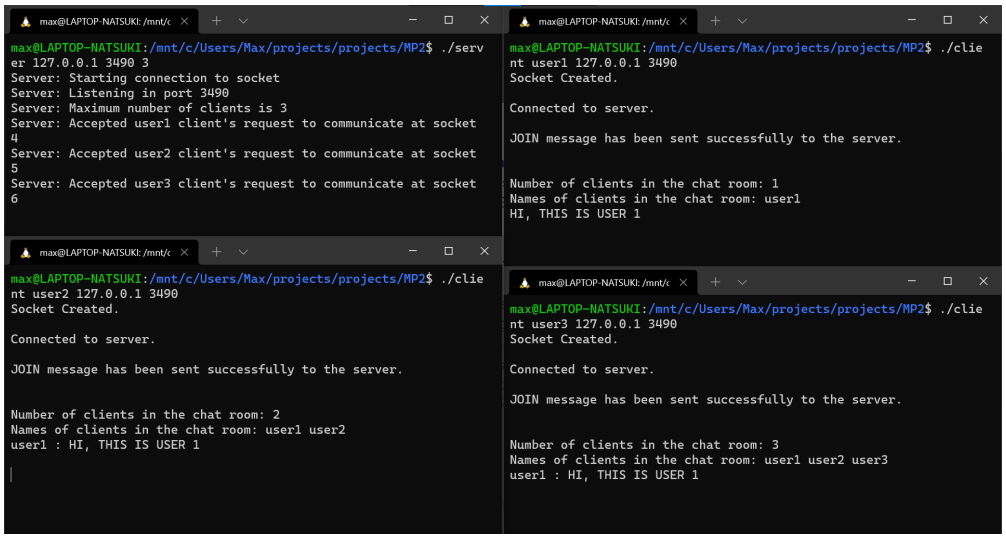
The LINK to the github repo is here: https://github.tamu.edu/baruah-dharmendra/ECEN602_Team04

Testcases:

Testcase 1:

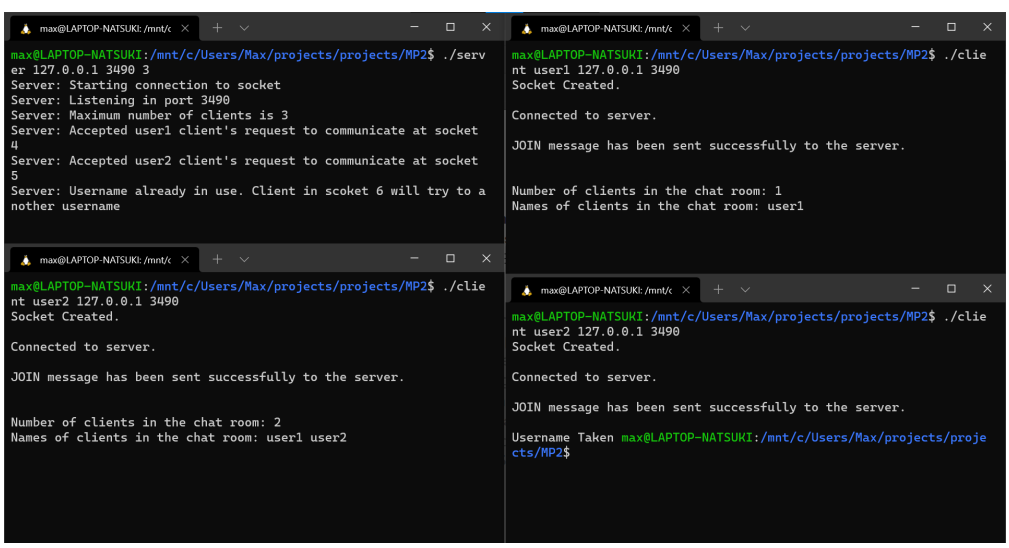
Normal operation of the chat client with three clients connected

For this use case, we have 3 clients (user1, user2, user3) in a group chat connected to the server.



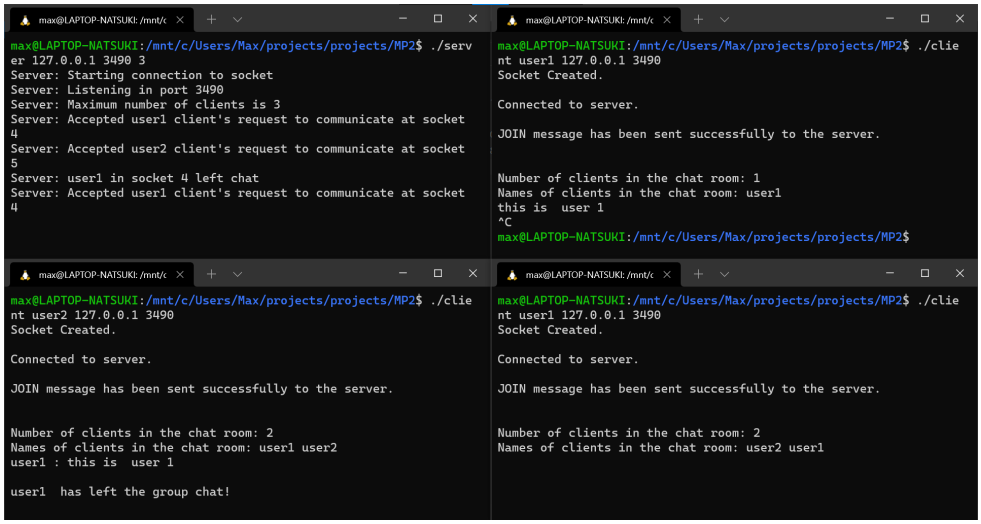
Testcase 2:

Server rejects client with duplicate username. For this use case, when the group chat already has user1 and user2, if a third user client with the name user1 tries to enter the chat, it results in an error as a client already exists with the same name.



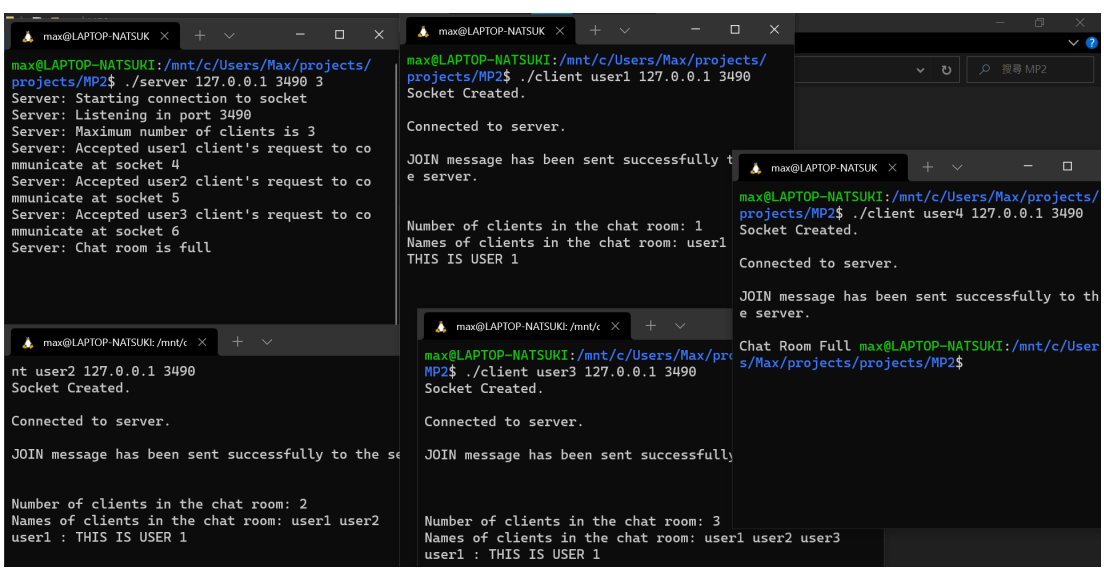
Testcase 3:

Server allows a previously used username to be reused In this use case, we had user3 exit the group chat and then have another client try to join the chat with the username user3.



Testcase 4:

Server rejects the client because it exceeds the maximum number of clients allowed. Here when we try to introduce a 4th client, the server rejects the client to enter the chat room as maximum client limit is reached.



<u>README.md</u>

TCP Simple Broadcast Chat Server and Client

Purpose:

This Project is developed as a part of Machine Problem 2 of Computer Networks and Communication course. It is performed as a team of two where we are supposed to implement a client and server for a simple chat service.

Implementation:

The Client Server model performs the following implementation:
1. Start the server first with the command line: server IPAdr Port users, where IPAdr is the IPv4 address of the server, Port is the port number on which the server is creating the chat room and users is how many clients are allow in the chat room. An instance of the server provides a single “chat room,” which can only handle a finite number of clients.
2. Start the client second with a command line: client client_name IPAdr Port, where client_name is the name of the client who is requesting access to the chat room of the server, IPAdr is the IPv4 address of the server and Port is the port number on which the server is listening. Clients must explicitly JOIN the session.
3. A client receives a list of the connected members of the chat session once they complete the JOIN transaction.
4. Clients use SEND messages to carry chat text, and clients receive chat text from the server using the FWD message.
5. Clients may exit unceremoniously at any time during the chat session.
6. The server detects a client exit, cleanup the resources allocated to that client and notify the other clients.

Running

Installation:

Clone this repository
```
git@github.tamu.edu:baruah-dharmendra/ECEN602_Team04.git
```

Building:

For this we will need standard C++ compiler installed in the machine in which the program is run. To build it can be directly done from the make file or individually by compiling the server client.

For building without the make file:

Build the client: ``` g++ -o Client client.cpp ```

Build the server: ``` g++ -o Server server.cpp ```

For building with the makefile we can just use the command
```
make all
```
Execution:

Open the terminal window and run: ``` ./Server 127.0.0.1 3490 3 ```

If the server response following should be visible in the terminal.
```
Server: Starting connection to socket
Server: Listening in port 3490
Server: Maximum number of client is 3

```
In another terminal run: ``` ./Client user1 127.0.0.1 3490```
for client 2 open another terminal and run: ``` ./Client user2 127.0.0.1 3490```

Depending on the number of users, we can open multiple clients.

The client 1 should respond to it with the following message:

```
Connected to server
JOIN message has been sent successfully to the server.

Number of clients in the chat room: 1
Names of clients in the chat room: user1

```

The client 2 should respond to it with the following message:

```
Connected to server
JOIN message has been sent successfully to the server.

Number of clients in the chat room: 2
Names of clients in the chat room: user1 user2
```

Once the connection to the client is established the server should show the following message:

```
Server: Accepted user1 client's request to communicate at socket 4
Server: Accepted user2 client's request to communicate at socket 5
```

Testing:

To test it, if at the client user1 terminal, following input is enter:

```
user1: HI, THIS IS USER 1
```

Then, the message will be BROADCASTED by the server and following message should be displayed in all the client side.

Exiting:

For closing the client control + C key can be used.

Following should appear at the client and the server side. In my case the user leaving the chat room is user1 and is connected to socket 4 of server.

Client terminal:

```
user 1 has left the group chat
```

Server terminal:

```
user1 in socket 4 left chat
```

```
---

## Test cases



Testcase 1:

Normal operation of the chat client with three clients connected

For this use case, we have 3 clients (user1, user2, user3) in a group chat connected to the server.


![](test_case_1.png)

Testcase 2:

Server rejects client with duplicate username
For this use case, when the group chat already has user1 and user2, if a third user client with the name user1 tries to enter the chat, it results in an error as a client already exists with the same name.

![](test_case_2.png)

Testcase 3:

Server allows a previously used username to be reused
In this use case, we had user3 exit the group chat and then have another client try to join the chat with the username user3.

![](test_case_3.png)

Testcase 4:

Server rejects the client because it exceeds the maximum number of clients allowed. Here when we try to introduce a 4th client, the server rejects the client to enter the chat room as maximum client limit is reached.

![](test_case_4.png)

---
```

Team

@dharmendrabaruah
@yehtungchi

Effort

The entire project was completed with equal efforts from either member of the team maintaining synergy. It was carried out in the university library, where both of the members were responsible for the analysis, coding, debugging, testing and documentation of the server client application.

Client Code:

```
#include <errno.h>

#include <sys/socket.h>

#include <stdio.h>

#include <string.h>

#include <arpa/inet.h>

#include <sys/types.h>

#include <stdlib.h>

#include <sys/select.h>

#include <unistd.h>

#include <netinet/in.h>

#include <sys/time.h>

#include <fcntl.h>

#include <netdb.h>


//Defining SBCP message header

struct sbcp_msg_header
```

```
{
  int vrsn;
  int type;
  int length;
};

//Defining SBCP Attribute
struct sbcp_attr
{
  int type;
  int length;
  char payload[512];
};

//Defining SBCP message
struct sbcp_msg
{
  struct sbcp_msg_header msg_header;
  struct sbcp_attr attr;
};

void set_msg_para(sbcp_msg msg, int H_vrsn, int H_type, int A_type, int
A_length){
      msg.msg_header.vrsn = H_vrsn;
  msg.msg_header.type = H_type;          //message type for join is 2
  msg.attr.type = A_type;                //attr type for username is 2
  msg.attr.length = A_length;
```

```
  // msg.attr.length = 2 + 2 + pl_len; //2bytes type, 2bytes length,
length of attr payload

}



//Defining JOIN fucntion

int join(char *pl, int c_sock)

{

  char packet[534];

  memset(packet, '\0', sizeof(packet));

  int pl_len = strlen(pl);



  struct sbcp_msg msg;

  set_msg_para(msg, 3, 2, 2, 2 + 2 + pl_len);//2bytes type, 2bytes length,
length of attr payload



  for (int i = 0; i < pl_len; i++)

  {

    msg.attr.payload[i] = pl[i];

  }

  msg.msg_header.length = 4 + 4 + pl_len; //4bytes versn type
length,4bytes sbcp_attr type length, pl



  sprintf(packet, "%d:%d:%d:%d:%d:%s", msg.msg_header.vrsn,
msg.msg_header.type, msg.msg_header.length, msg.attr.type,
msg.attr.length, msg.attr.payload);



  int sent = send(c_sock, packet, sizeof(packet), 0); //send message to
the socket

  if (sent == -1)

  {
```

```
    puts("Failed to send JOIN message to the server.\n");
    exit(-1);
  }
  puts("JOIN message has been sent successfully to the server.\n");
  return sent;
}

//Defining SEND MESSAGE fucntion
int send_msg(int c_sock, char *uname)
{
  char packet[534];
  char msg_send[512];
  memset(packet, '\0', sizeof(packet));
  memset(msg_send, '\0', sizeof(msg_send));
  fgets(msg_send, sizeof(msg_send), stdin);

  int msg_len = strlen(msg_send);

  struct sbcp_msg msg;
  set_msg_para(msg, 3, 4, 4, 2 + 2 + msg_len);//2bytes type, 2bytes
length, length of attr payload

  for (int i = 0; i < msg_len; i++)
  {
    msg.attr.payload[i] = msg_send[i];
  }
```

```
  msg.msg_header.length = 4 + 4 + msg_len; //4bytes versn type
length,4bytes sbcp_attr type length, pl



  //char sbcp_msg_send[1000];

  sprintf(packet, "%d:%d:%d:%d:%d:%s", msg.msg_header.vrsn,
msg.msg_header.type, msg.msg_header.length, msg.attr.type,
msg.attr.length, msg.attr.payload);



  int sent = send(c_sock, packet, sizeof(packet), 0); //send message to
the socket

  if (sent == -1)

  {

    puts("\nFailed to send message to the server.\n");

    return 0;

  }



  //puts("Message sent!");

  return sent;

}



//Defining RECV MESSAGE function

struct sbcp_msg recv_msg(char *recv_buff, int c, int readBytes)

{

  struct sbcp_msg msg;

  int rmsg_header_vrsn;

  int rmsg_header_type;

  int rmsg_header_len;

  int rmsg_attr_type;

  int rmsg_attr_len;
```

```
char rmsg_attr_payload[511];
//printf("raw: %s",recv_buff);
//Retrieve SBCP fields:

int field = 0;
char translate[readBytes];

for (int i = 0; i < readBytes; i++)
{
  if (field == 0)
  {
    if (recv_buff[i] == ':')
    { //header version
      msg.msg_header.vrsn = atoi(translate);
      memset(&translate, '\0', sizeof(translate) / sizeof(char));
      field++;
    }
    else
    {
      strncat(translate, &recv_buff[i], 1);
    }
  }
  else if (field == 1)
  {
    if (recv_buff[i] == ':')
    { //header type
      msg.msg_header.type = atoi(translate);
```

```
    memset(&translate, '\0', sizeof(translate) / sizeof(char));
    field++;
  }
  else
  {
    strncat(translate, &recv_buff[i], 1);
  }
}
else if (field == 2)
{
  if (recv_buff[i] == ':')
  { //header length
    msg.msg_header.length = atoi(translate);
    memset(&translate, '\0', sizeof(translate) / sizeof(char));
    field++;
  }
  else
  {
    strncat(translate, &recv_buff[i], 1);
  }
}
else if (field == 3)
{
  if (recv_buff[i] == ':')
  { //attribute type
    msg.attr.type = atoi(translate);
    memset(&translate, '\0', sizeof(translate) / sizeof(char));
```

```
      field++;
    }
    else
    {
      strncat(translate, &recv_buff[i], 1);
    }
  }
  else if (field == 4)
  {
    if (recv_buff[i] == ':')
    { //attribute length
      msg.attr.length = atoi(translate);
      memset(&translate, '\0', sizeof(translate) / sizeof(char));
      field++;
    }
    else
    {
      strncat(translate, &recv_buff[i], 1);
    }
  }
  else if (field == 5)
  {
    strncat(translate, &recv_buff[i], 1);
  }
}
strcpy(msg.attr.payload, translate); //attribute payload
```

```
  if (msg.attr.type == 3)

  {

    //Number of clients

    printf("\nNumber of clients in the chat room: %s\n",
msg.attr.payload);

    printf("Names of clients in the chat room: ");



    for (int i = 1; i <= atoi(msg.attr.payload); i++)

    {

      char recv_buf[534];

      int num = recv(c, recv_buf, 534, 0);

      recv_msg(recv_buf, c, num);

    }

    printf("\n");

  }

  else if (msg.attr.type == 2)

  {

    //Usernames

    printf("%s ", msg.attr.payload);

    //printf("Names of clients in the chat room: %s\n", msg.attr.payload);

  }

  else if (msg.attr.type == 4)

  {

    //Message

    char recv_buf[534];

    int num = recv(c, recv_buf, 534, 0);

    struct sbcp_msg ret_pack;

    ret_pack = recv_msg(recv_buf, c, num);
```

```
  printf(": %s\n", msg.attr.payload);
}
else if (msg.attr.type == 1)
{
  //Reason for failure
  if (strcmp(msg.attr.payload, "Abrupt Exit") == 0)
  {
    char recv_buf[534];
    int num = recv(c, recv_buf, 534, 0);
    struct sbcp_msg ret_pack;
    ret_pack = recv_msg(recv_buf, c, num);
    printf(" has left the group chat!\n");
  }
  else if (strcmp(msg.attr.payload, "Chat Room Full") == 0)
  {
    printf("%s ", msg.attr.payload);
    close(c);
    exit(-1);
  }
  else
  {
    printf("%s ", msg.attr.payload);
    close(c);
    exit(-1);
  }
}
```

```
  //printf("Payloads: %s\n", msg.attr.payload);

  return msg;

}


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

{


  if (argc < 4)

  { //make sure command line sets proper number of arguments

    printf("Please specify client's username, server's IPv4 address and
port number\n");

    return 0;

  }

  char uname[15];

  memset(uname, '\0', sizeof(uname));

  strcpy(uname, argv[1]);

  int ulen = strlen(uname);


  if (ulen > 16)

  { //make sure length of username is not more than 16

    printf("Please make sure username is not more than 16 characters.\n");

    return 0;

  }


  //Creating a socket

  int c = socket(AF_INET, SOCK_STREAM, 0);

  if (c == -1)
```

```
  {
    perror("The socket could not be created.\n");
    exit(-1);
  }
  puts("Socket Created.\n");

  char server_ip;
  int pnum;

  struct sockaddr_in serveraddr;
  memset(&serveraddr, '\0', sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  serveraddr.sin_port = htons(atoi(argv[3]));
  serveraddr.sin_addr.s_addr = inet_addr(argv[2]);

  //Conecting to server
  if (connect(c, (struct sockaddr *)&serveraddr, sizeof(serveraddr)) == -
1)
  {
    perror("Failed to connect to server");
    exit(-1);
  }
  puts("Connected to server. \n");

  int j_bytes = join(uname, c);
  char recv_buf[534];

  //I/O Multiplexing
```

```
int max_fd;
fd_set readfds;
FD_ZERO(&readfds);

while (1)
{
  FD_SET(0, &readfds); //For stdin input in readfds set
  FD_SET(c, &readfds); //For socket in readfds set

  max_fd = c;
  int rv = select(max_fd + 1, &readfds, NULL, NULL, NULL);
  if (rv == -1)
  {
    puts("Error in select.\n");
    exit(-1);
  }

  if (FD_ISSET(0, &readfds))
  {
    send_msg(c, uname);
  }
  else if (FD_ISSET(c, &readfds))
  {
    int num = recv(c, recv_buf, 534, 0);
    recv_msg(recv_buf, c, num);
  }
}
```

```
  //CLOSING CONNECTION

  puts("Client closing connection");

  close(c);

  return 0;

}
```

Server code:

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

//SBCP Packet Structure:

//Defining SBCP message header
struct sbcp_msg_header
{
  int vrsn; //3
  int type; // FWD (for server)
  int length; //SBCP message length
};

//Defining SBCP Attribute
struct sbcp_attr
{
  int type; //2 (username), 4 (message), 1 (Reason), or 3 (Client)
  int length; //length of SBCP attribute
  char payload[512]; //largest attribute type (message)
};

//Defining SBCP message
struct sbcp_msg
{
  struct sbcp_msg_header msg_header;
  struct sbcp_attr attr;
};
```

```
//Written Function (returns number of bytes written or -1 for error):
(FWD)
int written(int descriptor, struct sbcp_msg data){
	char packet[534]; //SBCP data packet
	memset(packet,'\0',sizeof(packet));

	//compress SBCP data into packet of type char:
	sprintf(packet, "%d:%d:%d:%d:%d:%s", data.msg_header.vrsn,
data.msg_header.type, data.msg_header.length, data.attr.type,
data.attr.length, data.attr.payload);
	int sent = send(descriptor, packet, 534, 0); //send message to the
socket

	if(sent == -1){ //if error in sending, return 0 bytes sent

		if (errno == EINTR){ //if process is interrupted, resend it
again:

			printf("Server: Write Interrupted. Rewriting Now.\n");
			written(descriptor, data);
		}

		perror("Server: Write Error");
		return -1;
	}

	return sent; //if transmitting message is successful, return number
of bytes sent
}

//Reading Function lets server read data received from client: (JOIN or
SEND)
struct sbcp_msg reading(int descriptor, char data[], int dataLength, int
*byteRD){
	//reads message from client
	int readBytes = recv(descriptor, data, dataLength, 0);
	//assign variable pointed by this pointer the number of bytes read
	*byteRD = readBytes;
	//temporary packet holder
	struct sbcp_msg packet;

	//Retrieve SBCP fields:
	int field = 0;
	char translate[readBytes];

	for (int i = 0; i < readBytes; i++){
		if (field == 0){
			//header version
			if (data[i] == ':'){
				packet.msg_header.vrsn = atoi(translate);
				memset(&translate, '\0',
sizeof(translate)/sizeof(char));
				field++;
			}
```

```
                else{
                        strncat(translate, &data[i], 1);
                }
        }
        else if (field == 1){
                //header type
                if (data[i] == ':'){
                        packet.msg_header.type = atoi(translate);
                        memset(&translate, '\0',
sizeof(translate)/sizeof(char));
                        field++;
                }
                else{
                        strncat(translate, &data[i], 1);
                }
        }
        else if (field == 2){
                //header length
                if (data[i] == ':'){
                        packet.msg_header.length = atoi(translate);
                        memset(&translate, '\0',
sizeof(translate)/sizeof(char));
                        field++;
                }
                else{
                        strncat(translate, &data[i], 1);
                }
        }
        else if (field == 3){
                //attribute type
                if (data[i] == ':'){
                        packet.attr.type = atoi(translate);
                        memset(&translate, '\0',
sizeof(translate)/sizeof(char));
                        field++;
                }
                else{
                        strncat(translate, &data[i], 1);
                }
        }
        else if (field == 4){
                //attribute length
                if (data[i] == ':'){
                        packet.attr.length = atoi(translate);
                        memset(&translate, '\0',
sizeof(translate)/sizeof(char));
                        field++;
                }
                else{
                        strncat(translate, &data[i], 1);
                }
        }
        else if (field == 5){
                strncat(translate, &data[i], 1);
```

```
            }
      }
      strcpy(packet.attr.payload, translate); //attribute payload

      if (packet.attr.length > 4){
            char tmp[packet.attr.length - 4];
            memset(&tmp, '\0', sizeof(tmp));
            for (int i = 0; i < packet.attr.length - 4; i++){
                  strncat(tmp, &packet.attr.payload[i], 1);
            }
            memset(&packet.attr.payload, '\0',
sizeof(packet.attr.payload));
            strcpy(packet.attr.payload, tmp);
      }

      if (readBytes == -1){ //if recv() returns -1, there is an error
            perror("Server: Read error so read again");
      }

      return packet; //return the SBCP packet
}

void sbcp_msg_para(sbcp_msg SBCP, int H_vrsn, int H_type, int A_type, int
A_length){
      SBCP.msg_header.vrsn = H_vrsn;
      SBCP.msg_header.type = H_type; //message type for FWD is 3
      SBCP.attr.type = A_type; //attr type for reason of failure is 1
      SBCP.attr.length = A_length;  //2bytes type, 2bytes length, length
of attr payload
      memset(SBCP.attr.payload, '\0',   sizeof(SBCP.attr.payload) /
sizeof(char));
      // strcpy(SBCP.attr.payload, err_msg);//
}

void failureToConnect(char messgae[], char err_msg[], sbcp_msg SBCP,
      int num_bytes, int new_sckt, int length){
      //reason of failure to connect, error message variable, SBCP,
      //number of bytes written to socket using written(), next socket
descriptor to connect to client

      //initialize message to be empty
      memset(&err_msg, '\0', sizeof(err_msg));
      //reason of failure to connect:
      strcpy(err_msg, messgae);

      //FWD reason of failure:
      sbcp_msg_para(SBCP, 3, 3, 1, length);// set parpmeters
      strcpy(SBCP.attr.payload, err_msg);

      SBCP.msg_header.length = 15 + length;
      //write back the same message to client
      num_bytes = written(new_sckt, SBCP);

      //Check for error in writing message to socket:
```

```
	if (num_bytes == -1){
		printf("Server: Error in forwarding message\n");
	}
}


int main(int argc, char *argv[]){ //command line: echos port (echos is
name of program, port is port number)
	//
	// declare varibles
	//
	//socket descriptor
	int sckt;
	//server socket address
	struct sockaddr_in my_addr;
	//client socket address wanting to connect in port
	struct sockaddr_in nxt_client;
	//next socket descriptor to connect to client
	int new_sckt;
	//message to be sent
	char msg[1024];

	//comparator to detect "no text" from client (detects whitespace)
	char emp[1024];
	memset(&emp,'\0', sizeof(emp));

	//comparator to detect "no text" from client (detects tab character)
	char emp1[1024];
	memset(&emp1,'\0', sizeof(emp1));

	//length of the message recieved by server
	int msgLength;
	//number of bytes written to socket using written()
	int num_bytes;
	//child process identification
	int pid;
	//size of socket address
	int sizeAddr = sizeof(struct sockaddr_in);

	int yes = 1;

	// list of master file descriptors
	fd_set master;
	// list of temporary file descriptors for select()
	fd_set temp;

	//maximum file descriptor number
	int fd_max;
	//timeout for waiting for I/O in select()
	int timeout = 10;
	int ready_fd;
	//maximum number of clients for chat room
	int max_clients = atoi(argv[3]);
	//number of clients currently in chat room
```

```
	int num_clients = 0;
	//array of usernames of clients
	char usernames [max_clients][16];
	//keeps track of index of each socket/fd
	int trace_id[max_clients];
	//SBCP data packet
	struct sbcp_msg SBCP;
	char usr_nm[16]; //current username of current client
	char err_msg[1024]; //error message to be sent
	struct sbcp_msg SBCP_cpy; //to copy SBCP packets

	//initialize sets of file descriptors:
	FD_ZERO(&master);
	FD_ZERO(&temp);

	//Establish socket descriptor:
	if ((sckt = socket(AF_INET, SOCK_STREAM, 0)) == -1){ //assign socket
descriptor
		perror("Server: socket"); // handling error
		exit(-1);
	}

	if (setsockopt(sckt, SOL_SOCKET, SO_REUSEADDR, &yes, sizeof(int)) ==
-1){ //reuse port if address is already in use
		perror("Server: setsockopt"); // handling error
		exit(-1);
	}

	//Get information from socket address:
	my_addr.sin_family = AF_INET; //host byte order

	if (argc < 2){ //make sure command line sets proper number of
arguments
		printf("Please specify the IP address, port number and maximum
clients allowed\n");
		return 0;
	}
	else{
		// short, network byte order (atoi to convert input argument
from char to integer)
		my_addr.sin_port = htons(atoi(argv[2]));
	}

	//assign server's IP with specified one in command line
	my_addr.sin_addr.s_addr = inet_addr(argv[1]);
	//clean lingering bytes used previously in address
	memset(&my_addr.sin_zero, '\0', 8);

	printf("Server: Starting connection to socket\n");

	//Bind:
	if (bind(sckt, (struct sockaddr*) &my_addr, sizeAddr) == -1){
		perror("Server: Bind Error");
		exit(-1);
```

```
	}

	//Listen:
	if (listen(sckt, max_clients) == -1){
		perror("Server: Listen Error");
		exit(-1);
	}

	printf("Server: Listening in port %s\n", argv[2]);

	printf("Server: Maximum number of clients is %d\n", max_clients);

	FD_SET(sckt, &master); //add listening socket to master set

	fd_max = sckt; //set maximum file descriptor number to listening
socket value

	//intialize usernames in chat room as nothing:
	for(int i = 0; i <= max_clients; i++){
		usernames[i][0] = '\0';
		trace_id[i] = sckt;
	}

	//Accept thread:
	while(1){
		FD_ZERO(&temp);
		temp = master; //update from master set

		ready_fd = select(fd_max+1, &temp, NULL, NULL, NULL);
//select() waits for multiple inputs/outputs from clients
		if (ready_fd == -1){
			perror("Server: Select Error");
			exit(1);
		}

		for (int fnum = 0; fnum <= fd_max; fnum++){ //current file
descriptor
			if(FD_ISSET(fnum, &temp)){
				if(fnum == sckt){ //if wanting to connect, JOIN:

					//accept clients wanting to connect:

					if ((new_sckt = accept(sckt, (struct
sockaddr*) &nxt_client, (socklen_t*) &sizeAddr)) == -1){
						perror("Server: Accept Error");
						continue;
					}

					//accept only if max number of clients is not
reached:
					if (abs(max_clients-num_clients) != 0) {

						//get JOIN message from client:
```

```
                                    //get username of client:
                                    memset(&msg, '\0', sizeof(msg));
//initialize username to be empty
                                    SBCP = reading(new_sckt, msg, 1024,
&msgLength); //read username from client

                                    if ((&msg[msgLength-1] == NULL) ||
(msgLength <= 0)){ //if last character inputed indicates EOF, it means
client is disconnected
                                          printf("Server: client wanting to
connect in socket %d failed to connect\n", new_sckt);
                                          close(new_sckt); //close socket
                                    }

                                    //check if SBCP is JOIN message and
username attribute:
                                    if (SBCP.attr.type == 2 &&
SBCP.msg_header.type == 2 && msgLength != -1){
                                          strncpy(usr_nm, SBCP.attr.payload,
15);
                                          usr_nm[SBCP.attr.length] = '\0';
//clear remaining whitespace
                                    }
                                    else{ //Reason of Failure (no JOIN
message recieved)
                                          //FWD reason of failure to
connect:

                                          //FWD reason of failure:
                                          failureToConnect("JOIN MSG
Failed", err_msg, SBCP, num_bytes, new_sckt, 15);


                                          printf("Server: Not recieved JOIN
message from client in socket %d\n", new_sckt);
                                          memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message buffer
                                          close(new_sckt); //Disconnect
                                          continue;
                                    }

                                    //compare this username from the
usernames in chat room:
                                    int in_use = 0;
                                    for (int i = 0; i < num_clients; i++){
                                          if (strcmp(usr_nm,usernames[i]) ==
0){
                                                in_use = 1;
                                          }
                                    }
                                    //if username already in use, tell
client it is already in use
                                    if(in_use){
```

```
                                        //FWD reason of failure:
                                        failureToConnect("Username Taken",
err_msg, SBCP, num_bytes, new_sckt, 14);

                                        printf("Server: Username already
in use. Client in scoket %d will try to another username\n", new_sckt);
                                        memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message buffer
                                        close(new_sckt); //Disconnect
                                        continue;
                                  }

                                  //add new username to chat room:
                                  strcpy(usernames[num_clients], usr_nm);
                                  //keep track of which client/socket has
what username:
                                  trace_id[num_clients] = new_sckt;

                                  printf("Server: Accepted %s client's
request to communicate at socket %d\n", usernames[num_clients], new_sckt);

                                  FD_SET(new_sckt, &master); //push in
new socket to master set
                                  num_clients++; //increment number of
clients

                                  //update maximum value of file
descriptor in master set:
                                  if (new_sckt > fd_max){
                                        fd_max = new_sckt;
                                  }

                                  //FWD ACK to client:
                                  //Number of clients in chat room:
                                  //SBCP attribute 'Client Count':

                                  // set parameters: message type for FWD
is 3, attr type for client count is 3
                                  sbcp_msg_para(SBCP, 3, 3, 3,
(sizeof(num_clients)/sizeof(int)));

                                  sprintf(SBCP.attr.payload, "%d",
num_clients);

                                  SBCP.msg_header.length = 15 +
(sizeof(num_clients)/sizeof(int));
                                  //write back the same message to client
                                  num_bytes = written(new_sckt, SBCP);

                                  //Check for error in writing message to
socket:
                                  if (num_bytes == -1){
                                        printf("Server: Error in
forwarding message\n");
```

```
                                    }

                                    //Usernames of clients in chat room:
                                    //SBCP attribute 'Username':

                                    for(int n = 0; n < num_clients; n++){
                                          // set parameters: message type
for FWD is 3, attr type for username is 2
                                          sbcp_msg_para(SBCP, 3, 3, 2,
(sizeof(num_clients)/sizeof(int)));

                                          strcpy(SBCP.attr.payload,
usernames[n]);

                                          SBCP.msg_header.length = 15 +
(sizeof(usernames[n])/sizeof(char));
                                          num_bytes = written(new_sckt,
SBCP); //write back the same message to client

                                          //Check for error in writing
message to socket:
                                          if (num_bytes == -1){
                                                printf("Server: Error in
forwarding message\n");
                                          }
                                    }

                              }
                              else{ //if max number of clients is reached,
close/recycle socket

                                    //FWD reason of failure:
                                    failureToConnect("Chat Room is Full",
err_msg, SBCP, num_bytes, new_sckt, 14);

                                    printf("Server: Chat room is full\n");
                                    memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message buffer
                                    close(new_sckt); //Disconnect
                              }
                        }
                        else{ //SEND AND FWD

                              //Find username of this client:
                              int indices;
                              char username[16];
                              for (int i = 0; i < num_clients; i++){
                                    if (fnum == trace_id[i]){
                                          strcpy(username, usernames[i]);
                                          indices = i;
                                          break;
                                    }
                              }
```

```
                            //Read message SEND by client:
                            memset(&msg, '\0', sizeof(msg)); //initialize
message to be empty

                            //process SBCP SEND() from client:
                            SBCP = reading(fnum, msg, 1024, &msgLength);
//read username from client

                            //check if SBCP is SEND message and message
attribute:
                            if (SBCP.msg_header.type != 4 || msgLength ==
-1){

                                  //FWD reason of failure:
                                  failureToConnect("SEND MSG Failed",
err_msg, SBCP, num_bytes, new_sckt, 15);

                                  memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message buffer
                                  printf("Server: Not recieved SEND
message from client in socket %d\n", new_sckt);
                            }

                            if ((&msg[msgLength-1] == NULL) || (msgLength
<= 0)){ //if last character inputed indicates EOF, it means client is
disconnected
                                  printf("Server: %s in socket %d left
chat\n", username, fnum);

                                  //broadcast exit of client:
                                  memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message
                                  strcpy(err_msg, "Abrupt Exit");
      //reason of failure

                                  for (int i = 0; i <= fd_max; i++){
                                        if (FD_ISSET(i, &master) && i !=
sckt && i != fnum){
                                              //FWD reason of failure
message with the username in 2 packets:

                                              //FWD reason of failure:
                                              //set parameters: message
type for FWD is 3, attr type for reason of failure is 1, length of attr
payload
                                              sbcp_msg_para(SBCP, 3, 3, 1,
11);

                                              strcpy(SBCP.attr.payload,
err_msg);

                                              SBCP.msg_header.length = 15
+ 11;
```

```
                                                num_bytes = written(i,
SBCP); //write back the same message to client

                                                //Check for error in writing
message to socket:
                                                if (num_bytes == -1){
                                                      printf("Server: Error
in forwarding message\n");
                                                }

                                                //FWD username:
                                                //set parameters: message
type for FWD is 3, attr type for username is 2, length of attr payload
                                                sbcp_msg_para(SBCP, 3, 3, 2,
(sizeof(username)/sizeof(char)));

                                                strcpy(SBCP.attr.payload,
username);

                                                SBCP.msg_header.length = 15
+ (sizeof(username)/sizeof(char));

                                                num_bytes = written(i,
SBCP); //write back the same message to client

                                                //Check for error in writing
message to socket:
                                                if (num_bytes == -1){
                                                      printf("Server: Error
in forwarding message\n");
                                                }
                                          }
                                    }
                                    memset(&err_msg, '\0',
sizeof(err_msg)); //clear/reset message

                                    //recycle resources reserved for this
client:
                                    close(fnum); //close socket
                                    FD_CLR(fnum, &master); //remove socket
from master set
                                    num_clients--; //decrement number of
clients

                                    //delete username:
                                    if (indices < max_clients -1){ //if not
last client, delete from the middle and copy over the remaining clients:

                                          //copy over remianing usernames
and their indices:

                                          for (int i = indices; i <
max_clients - 1; i++){
```

```
                                                strcpy(usernames[i],
usernames[i+1]);
                                                trace_id[i] = trace_id[i+1];
                                          }

                                          //erase redudancy:

                                          for (int i = num_clients; i <
max_clients; i++){
                                                memset(&usernames[i], '\0',
sizeof(usernames[i])/sizeof(char));
                                                trace_id[i] = sckt;
                                          }
                                    }
                                    else{ //otherwise, just delete the last
client
                                          memset(&usernames[indices], '\0',
sizeof(usernames[indices]));
                                          trace_id[indices] = sckt;
                                    }

                                    continue;
                              }

                              memset(&emp,' ', msgLength-1); //set
comparator as whitespace of the same length of message except the
termination ('\n')
                              memset(&emp1,'\t', msgLength-1); //set
comparator as tab of the same length of message except the termination
('\n')

                              if ((strcmp(msg,emp) != 0) &&
(strcmp(msg,emp1) != 0)){ //do not echo when no text/character is sent
(whitespace is not considered)
                                    //copy SEND message packet
                                    SBCP_cpy = SBCP;
                                    SBCP.msg_header.vrsn = 3;
                                    SBCP.msg_header.type = 3; //message
type for FWD is 3

                                    //create SBCP username packet:
                                    //set parameters: message type for FWD
is 3, attr type for username is 2, length of attr payload
                                    sbcp_msg_para(SBCP, 3, 3, 2,
(sizeof(username)/sizeof(char)));

                                    strcpy(SBCP.attr.payload, username);

                                    SBCP.msg_header.length = 15 +
(sizeof(username)/sizeof(char));

                                    for (int i = 0; i <= fd_max; i++){
                                          if (FD_ISSET(i, &master) && i !=
sckt && i != fnum){
```

```
                                                //FWD message with the
username in 2 packets:

                                                //FWD message packet:
                                                num_bytes = written(i,
SBCP_cpy); //write back the same message to client

                                                //Check for error in writing
message to socket:
                                                if (num_bytes == -1){
                                                      printf("Server: Error
in forwarding message\n");
                                                }

                                                //FWD username packet:
                                                num_bytes = written(i,
SBCP); //write back the same message to client

                                                //Check for error in writing
message to socket:
                                                if (num_bytes == -1){
                                                      printf("Server: Error
in forwarding message\n");
                                                }

                                          }
                                    }
                              }
                              else{
                                    //if blank, send nothing
                                    strcpy(msg, "if blank, send nothing");
                                    //set parameters: message type for FWD
is 3, attr type for blank is 4, length of attr payload
                                    sbcp_msg_para(SBCP, 3, 3, 4, 0);

                                    strcpy(SBCP.attr.payload, msg);
                                    SBCP.msg_header.length = 15;

                                    //copy SEND message packet
                                    SBCP_cpy = SBCP;

                                    //create SBCP username packet:
                                    //set parameters: message type for FWD
is 3, attr type for username is 2, length of attr payload
                                    sbcp_msg_para(SBCP, 3, 3, 2,
(sizeof(username)/sizeof(char)));

                                    strcpy(SBCP.attr.payload, username);

                                    SBCP.msg_header.length = 15 +
(sizeof(username)/sizeof(char));

                                    for (int i = 0; i <= fd_max; i++){
```

```
                                        if (FD_ISSET(i, &master) && i !=
sckt && i != fnum){
                                              //FWD message with the
username in 2 packets:

                                              //FWD message:

                                              num_bytes = written(i,
SBCP_cpy); //write back the same message to client

                                              //Check for error in writing
message to socket:
                                              if (num_bytes == -1){
                                                    printf("Server: Error
in forwarding message\n");
                                              }

                                              //FWD username:
                                              num_bytes = written(i,
SBCP); //write back the same message to client

                                              //Check for error in writing
message to socket:
                                              if (num_bytes == -1){
                                                    printf("Server: Error
in forwarding message\n");
                                              }

                                        }
                                  }
                            }

                            memset(&msg, '\0', sizeof(msg));
//clear/reset message
                            memset(&emp,'\0', sizeof(emp));
memset(&emp1,'\0', sizeof(emp1));//clear/reset comparators
                      }
                }
            }
      }
      return 0;
}
```