Project 4

Network Programming

Alex Benasutti and Alex Van Orden

CSC345-01

December 2 2019

**C Code (main\_server.c):**

/\*

\* Alex Benasutti and Alex Van Orden

\* CSC345-01

\* Network Programming

\* main\_server.c

\*/

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <pthread.h>

#include <time.h>

#define PORT\_NUM 1004

**#define ROOM\_MAX 255**

**int room\_tot = 0; // Integer to hold the total amount of available rooms**

**int room\_list[ROOM\_MAX+1] = {0}; // Array to hold the amount of people in each room**

void error(const char \*msg)

{

perror(msg);

exit(1);

}

/\* Declare user structure \*/

typedef struct \_USR {

int clisockfd; // socket file descriptor

**char username[12]; // user name**

**int room;**

**int color;**

struct \_USR\* next; // for linked list queue

} USR;

/\* Allow linked lists of users for user traversal \*/

USR \*head = NULL; // traversal

USR \*tail = NULL; // adding new client

**void print\_client\_list();**

int choose\_color();

/\* add\_tail - Add a new client/user to the end of the linked list \*/

void add\_tail(int newclisockfd, char\* username, int roomNum)

{

printf("Adding new user to tail\n");

if (head == NULL) {

// printf("Head is NULL\n");

head = (USR\*) malloc(sizeof(USR));

head->clisockfd = newclisockfd;

// printf("Adding attributes\n");

**strcpy(head->username, username);**

**head->room = roomNum;**

**head->color = choose\_color();**

// printf("making tail head\n");

head->next = NULL;

tail = head;

// printf("New user - %s\n", tail->username);

} else {

// printf("Head is not NULL\n");

tail->next = (USR\*) malloc(sizeof(USR));

tail = tail->next;

tail->clisockfd = newclisockfd;

**strcpy(tail->username, username);**

**tail->room = roomNum;**

**tail->color = choose\_color();**

tail->next = NULL;

// printf("New user - %s\n", tail->username);

}

}

**int choose\_color()**

**{**

**int clr\_idx\_1 = rand() % 7;**

**// int clr\_idx\_2 = rand() % 16;**

**// char\* color;**

**// sprintf(color, "\x1B[%dm", clr\_idx\_1\*16+clr\_idx\_2);**

**// int color = clr\_idx\_1\*16+clr\_idx\_2;**

**int color = clr\_idx\_1 + 31;**

**// printf("COLOR: %d\n", color);**

**return color;**

**}**

**/\* broadcast - send message from client to all clients \*/**

**void broadcast(int fromfd, char\* message)**

**{**

**// printf("Attempting to broadcast\n");**

**// figure out sender address**

**struct sockaddr\_in cliaddr;**

**socklen\_t clen = sizeof(cliaddr);**

**if (getpeername(fromfd, (struct sockaddr\*)&cliaddr, &clen) < 0) error("ERROR Unknown sender!");**

**char sendername[12];**

**int senderroom;**

**int sendercolor;**

**USR\* cur = head;**

**/\* Loop to grab username of sender by matching file descriptors \*/**

**while (cur != NULL)**

**{**

**if(cur->clisockfd == fromfd) // looking for sender**

**{**

**strcpy(sendername, cur->username);**

**senderroom = cur->room;**

**sendercolor = cur->color;**

**// if(strlcpy(username, cur->username, 12) >= 12)**

**// {**

**// printf("Warning: %s concatenated to %s\n", cur->username, username);**

**// }**

**break;**

**}**

**cur = cur->next;**

**}**

**// printf("Got sender: %s\n", sendername);**

**// traverse through all connected clients**

**cur = head;**

**while (cur != NULL) {**

**// check if cur is not the one who sent the message**

**if (cur->clisockfd != fromfd && cur->room == senderroom) {**

**// printf("Printing the message now\n");**

**char buffer[512];**

**memset(buffer, 0, 512);**

**// prepare message**

**sprintf(buffer, "\x1B[%dm[%d][%s (%s)]:\x1B[0m %s", sendercolor, senderroom, inet\_ntoa(cliaddr.sin\_addr), sendername, message);**

**// sprintf(buffer, "[%s]:%s", username, message);**

**int nmsg = strlen(buffer);**

**// send!**

**// printf("Sending the message now\n");**

**int nsen = send(cur->clisockfd, buffer, nmsg, 0);**

**if (nsen != nmsg) error("ERROR send() failed");**

**}**

**cur = cur->next;**

**}**

**// printf("Message finished printing\n");**

**}**

**void print\_client\_list()**

**{**

**USR\* cur = head;**

**if(head != NULL)**

**{**

**printf("Current Connected Clients\n");**

**cur = head;**

**while (cur != NULL)**

**{**

**printf("%s\n", cur->username);**

**cur = cur->next;**

**}**

**}**

**else**

**{**

**printf("No Connected Clients\n");**

**}**

**}**

/\* ThreadArgs - Holds client socket file descriptor \*/

typedef struct \_ThreadArgs {

int clisockfd;

} ThreadArgs;

/\* thread\_main - main worker function for each thread \*/

void\* thread\_main(void\* args)

{

// make sure thread resources are deallocated upon return

pthread\_detach(pthread\_self());

// get socket descriptor from argument

int clisockfd = ((ThreadArgs\*) args)->clisockfd;

free(args);

//-------------------------------

// Now, we receive/send messages

char buffer[256];

memset(buffer, 0, 256);

int nsen, nrcv;

USR\* cur = head;

USR\* prv = NULL;

memset(buffer, 0, 256);

// printf("Waiting to receive first buffer\n");

nrcv = recv(clisockfd, buffer, 255, 0);

// printf("Received first buffer\n");

// Very first buffer receieve

if (nrcv < 0)

{

error("ERROR recv() failed");

}

else if (nrcv == 0) // when a client disconnects

{

// printf("Client disconnected, removing client fd\n");

// remove file descriptor to disconnected client

while (cur != NULL)

{

if(cur->clisockfd == clisockfd)

{

**printf("Disconnected Client: (%s)\n", cur->username);**

**room\_list[(cur->room)- 1]--;**

if(prv == NULL)

{

// printf("previous is null\n");

head = cur->next; // change head

free(cur); // free old head

}

else // some node other than the head needs to be removed

{

// printf("%s links to %s\n",prv->next->username, cur->next->username);

prv->next = cur->next;

free(cur);

}

break;

}

prv = cur;

cur = cur->next;

}

}

**while (nrcv > 0) {**

**// we send the message to everyone except the sender**

**broadcast(clisockfd, buffer);**

memset(buffer, 0, 256);

// printf("Check for new messages\n");

nrcv = recv(clisockfd, buffer, 255, 0);

// printf("New buffer received: %s\n", buffer);

if (nrcv < 0) error("ERROR recv() failed");

else if (nrcv == 0) // when a client disconnects

{

//remove file discriptor to disconected client

while (cur != NULL)

{

if(cur->clisockfd == clisockfd)

{

**printf("Disconnected Client: (%s)\n", cur->username);**

// printf("Previous List\n");

// print\_client\_list();

// printf("cur->room: %d\n", cur->room);

// printf("room\_list[(cur->room)- 1]: %d\n", room\_list[(cur->room)- 1]);

**room\_list[(cur->room)- 1]--;**

if(prv == NULL)

{

// printf("hmm previous is null\n");

head = cur->next;

free(cur);

}

else

{

// printf("%s links to %s\n",prv->next->username, cur->next->username);

prv->next = cur->next;

free(cur);

}

break;

}

prv = cur;

cur = cur->next;

}

}

}

close(clisockfd);

//-------------------------------

return NULL;

}

**int getRoomNum(int clisockfd, struct sockaddr\_in cli\_addr, char\* room)**

**{**

**int digitFlag = 1;**

**int noFlag = 0;**

**int roomNum;**

**/\* Confirm whether every part of the message, besides \n, was an integer \*/**

**for (int i=0;i<strlen(room)-1;i++)**

**{**

**if (room[i] < 48 || room[i] > 57) // chars 0 to 9 are 48 to 57**

**{**

**digitFlag = 0;**

**break;**

**}**

**}**

**// check if room was submitted as nothing**

**if (room[0] == 10)**

**{**

**digitFlag = 0;**

**noFlag = 1;**

**}**

**// room input is an integer room, check that it exists and return if true**

**if (digitFlag)**

**{**

**printf("Digit room specified\n");**

**roomNum = atoi(room);**

**printf("roomNum: %d\n", roomNum);**

**// Check to see if room num is valid**

**if (roomNum > 0 && roomNum <= ROOM\_MAX && roomNum <= room\_tot)**

**{**

**memset(room,0,256);**

**sprintf(room,"Success\n");**

**// Send junk array to client**

**int nmsg = strlen(room);**

**int nsen = send(clisockfd, room, nmsg, 0);**

**if (nsen != nmsg) error("ERROR send() failed");**

**return roomNum;**

**}**

**else // punish the client if invalid**

**{**

**memset(room,0,256);**

**sprintf(room,"Well this is unfortunate\n");**

**// Send junk array to client**

**int nmsg = strlen(room);**

**int nsen = send(clisockfd, room, nmsg, 0);**

**if (nsen != nmsg) error("ERROR send() failed");**

**return 0;**

**}**

**// else error("ERROR room number out of range or nonexistent");**

**}**

**// room input is a string, check that it = "new" and return new room number if true, broadcast to client**

**else**

**{**

**int newFlag = strcmp(room,"new");**

**// client asks for new room, create it**

**if ((newFlag == 0 && room\_tot < ROOM\_MAX) || (noFlag == 1 && room\_tot == 0))**

**{**

**printf("Creating new room\n");**

**room\_tot += 1;**

**roomNum = room\_tot;**

**// Send message of new chat room back to client**

**char buffer[256];**

**memset(buffer, 0, 256);**

**sprintf(buffer, "Connected to %s with new room number: %d\n", inet\_ntoa(cli\_addr.sin\_addr), roomNum);**

**int nmsg = strlen(buffer);**

**int nsen = send(clisockfd, buffer, nmsg, 0);**

**if (nsen != nmsg) error("ERROR send() failed");**

**// Clear this again because we were having text problems**

**memset(buffer, 0, 256);**

**return roomNum;**

**}**

**else if (noFlag == 1 && room\_tot != 0 && room\_tot < ROOM\_MAX) // client input nothing, prompt client to choose a room from a list**

**{**

**printf("No room specified from client\n");**

**// Time to prompt user to choose a room**

**int maxLen = 4096;**

**// Create buffer array of all possible choices to send to client**

**char buffer[4096];**

**memset(buffer, 0, 4096);**

**// Start putting stuff in the buffer array**

**sprintf(buffer, "Server says the following options are available:\n");**

**for (int i=0;i<room\_tot;i++)**

**{**

**char room[32];**

**int people = room\_list[i];**

**// Create a string for each room**

**if (people > 1 || people == 0) sprintf(room, "\tRoom %d: %d people\n", i+1, people);**

**else sprintf(room, "\tRoom %d: %d person\n", i+1, people);**

**// Add room string to buffer array**

**if (strlen(buffer) + strlen(room) < maxLen) strcat(buffer, room);**

**else error("ERROR max length of room decision string exceeded");**

**}**

**strcat(buffer, "Choose a room number or type [new] to create a new room: ");**

**// Send prompt to client and let them choose a room**

**int nmsg = strlen(buffer);**

**int nsen = send(clisockfd, buffer, nmsg, 0);**

**if (nsen != nmsg) error("ERROR send() failed");**

**// Get new input from client**

**char newbuf[256];**

**memset(newbuf, 0, 256);**

**int nrcv = recv(clisockfd, newbuf, 255, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**// Get room num using a new call to the function**

**return getRoomNum(clisockfd, cli\_addr, newbuf);**

**}**

**}**

**return 0;**

**}**

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

{

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0) error("ERROR opening socket");

struct sockaddr\_in serv\_addr;

socklen\_t slen = sizeof(serv\_addr);

memset((char\*) &serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = INADDR\_ANY;

//serv\_addr.sin\_addr.s\_addr = inet\_addr("192.168.1.171");

serv\_addr.sin\_port = htons(PORT\_NUM);

int status = bind(sockfd,

(struct sockaddr\*) &serv\_addr, slen);

if (status < 0) error("ERROR on binding");

listen(sockfd, 5); // maximum number of connections = 5

/\* Main loop for accepting clients and sending messages \*/

while(1)

{

// Accept a new client into the server

struct sockaddr\_in cli\_addr;

socklen\_t clen = sizeof(cli\_addr);

int newsockfd = accept(sockfd, (struct sockaddr \*) &cli\_addr, &clen);

if (newsockfd < 0) error("ERROR on accept");

**// Temporarily link with just-created client to retrieve room number**

**char room[256];**

**// Clear array for new room number and receive room number from client**

**memset(room, 0, 256);**

**int nrcv = recv(newsockfd, room, 255, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**// Get either a new or existing ROOM NUMBER**

**int roomNum = getRoomNum(newsockfd, cli\_addr, room);**

**// if (roomNum == 0) error("ERROR getRoomNum() failed");**

**// getRoomNum failed - punish the client, not the server**

**if (roomNum == 0)**

**{**

**printf("getRoomNum() failed oh no\n");**

**continue;**

**}**

**// increment number of people in room**

**room\_list[roomNum-1] += 1;**

**// Temporarily link with just-created client to retrieve the user name**

**char username[256];**

**// Clear array for new username**

**memset(username, 0, 256);**

**nrcv = recv(newsockfd, username, 255, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**printf("Connected: %s (%s)\n", inet\_ntoa(cli\_addr.sin\_addr), username);**

add\_tail(newsockfd, username, roomNum); // add this new client to the client list

// print client list if thread creation was successful

// print\_client\_list();

// prepare ThreadArgs structure to pass client socket

ThreadArgs\* args = (ThreadArgs\*) malloc(sizeof(ThreadArgs));

if (args == NULL) error("ERROR creating thread argument");

args->clisockfd = newsockfd;

**pthread\_t tid;**

**if (pthread\_create(&tid, NULL, thread\_main, (void\*) args) != 0) error("ERROR creating a new thread");**

**else print\_client\_list();**

}

close(sockfd);

return 0;

}

**C Code (main\_client.c):**

/\*

\* Alex Benasutti and Alex Van Orden

\* CSC345-01

\* Network Programming

\* main\_client.c

\*/

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <netdb.h>

#include <pthread.h>

#define PORT\_NUM 1004

**#define ROOM\_MAX 255**

**int room\_tot = 0;**

void error(const char \*msg)

{

perror(msg);

exit(0);

}

typedef struct \_ThreadArgs {

int clisockfd;

} ThreadArgs;

void\* thread\_main\_recv(void\* args)

{

// Commented the detach out for now as is doesn't seem to affect anything

if (pthread\_detach(pthread\_self()) != 0)

error("ERROR pthread\_detach() (recv) failed");

int sockfd = ((ThreadArgs\*) args)->clisockfd;

free(args);

// keep receiving and displaying message from server

char buffer[512];

int n;

n = recv(sockfd, buffer, 512, 0);

printf("\n%s\n", buffer);

while (n > 0) {

printf("Inside recv loop...\n");

memset(buffer, 0, 512);

n = recv(sockfd, buffer, 512, 0);

if (n < 0) error("ERROR recv() failed");

printf("\n%s\n", buffer);

}

printf("Exiting recv loop...\n");

return NULL;

}

void\* thread\_main\_send(void\* args)

{

//if (pthread\_detach(pthread\_self()) != 0)

//error ("ERROR pthread\_detach() (send) failed");

int sockfd = ((ThreadArgs\*) args)->clisockfd;

free(args);

// keep sending messages to the server

char buffer[256];

int n;

**int username\_specified = 0;**

while (1)

{

printf("Inside send loop...\n");

// You will need a bit of control on your terminal

// console or GUI to have a nice input window.

**if (!username\_specified) printf("\nEnter a username: ");**

else printf("\nEnter a message: ");

memset(buffer, 0, 256);

fgets(buffer, 255, stdin);

if (strlen(buffer) == 1) buffer[0] = '\0';

// Demolish that newline character \n

if (buffer[strlen(buffer)-1] == 10) buffer[strlen(buffer)-1] = '\0';

**// Assure that the user-specified username is 12 characters or less**

**if (!username\_specified && strlen(buffer) > 12) continue;**

**else username\_specified = 1**;

n = send(sockfd, buffer, strlen(buffer), 0);

if (n < 0) error("ERROR writing to socket");

if (n == 0) break; // we stop transmission when user types empty string

}

printf("Exiting send loop...\n");

return NULL;

}

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

{

// argv[2] = room number or new

// usage: ./main\_client\_full ip-address-of-server username

// NOTE: This is different way from what's described in pdf

// must follow the description in pdf instead of this following implementation

if (argc < 2) error("Please specify hostname");

int sockfd = socket(AF\_INET, SOCK\_STREAM, 0);

if (sockfd < 0) error("ERROR opening socket");

struct sockaddr\_in serv\_addr;

socklen\_t slen = sizeof(serv\_addr);

memset((char\*) &serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr(argv[1]);

serv\_addr.sin\_port = htons(PORT\_NUM);

printf("Trying to connect to %s...\n", inet\_ntoa(serv\_addr.sin\_addr));

int status = connect(sockfd,

(struct sockaddr \*) &serv\_addr, slen);

if (status < 0) error("ERROR connecting");

**// see if the room number entered as an arg is a digit**

**int digitFlag = 1;**

**// send room number to server**

**char buffer[256];**

**// clear buffer**

**memset(buffer, 0, 256);**

**// set room number into buffer**

**if (argc > 2) strcpy(buffer, argv[2]);**

**else**

**{**

**buffer[0] = 10;**

**digitFlag = 0;**

**}**

**// Error handling on bad room number arguments**

**/\* Confirm whether every part of the message, besides \n, was an integer \*/**

**for (int i=0;i<strlen(buffer)-1;i++)**

**{**

**if (buffer[i] < 48 || buffer[i] > 57) // chars 0 to 9 are 48 to 57**

**{**

**digitFlag = 0;**

**break;**

**}**

**}**

**// If client entered a string and it's not "new" punish them**

**if (argc > 2 && digitFlag == 0 && strcmp(buffer,"new") != 0) error("ERROR Invalid new");**

**// send room number to server**

**status = send(sockfd, buffer, strlen(buffer), 0);**

**if (status < 0) error("ERROR writing to socket");**

**// make sure the room NUMBER you sent is valid**

**if (digitFlag)**

**{**

**printf("You entered a digit\n");**

**memset(buffer, 0, 256);**

**int nrcv = recv(sockfd, buffer, 255, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**printf("\n%s", buffer);**

**// if unsuccessful in finding a valid room, kill the client**

**if (strcmp(buffer,"Success\n") != 0) error("ERROR Invalid room number");**

**}**

**// print out new room number by receiving a room count from the server**

**if (strcmp(buffer,"new") == 0)**

**{**

**printf("Client is getting a new room\n");**

**// Client receives a new room number and is announced through this recv()**

**memset(buffer, 0, 256);**

**int nrcv = recv(sockfd, buffer, 255, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**printf("\n%s\n", buffer);**

**}**

**else if (argc < 3) // "new" keyboard nor room number specified, prompt client to choose room OR automatically connect if this is first room created**

**{**

**printf("Client must be prompted or is getting first room\n");**

**char bigbuf[4096];**

**memset(bigbuf, 0, 4096);**

**int nrcv = recv(sockfd, bigbuf, 4096, 0);**

**if (nrcv < 0) error("ERROR recv() failed");**

**printf("\n%s\n", bigbuf);**

**char word[9]; // will hold the word "Connected" - the first word of a message that prints when you automatically join a room**

**strncpy(word, bigbuf, 9);**

**//printf("This is word: %s", word);**

**// Checking the contents of the output buffer is the easiest way I can check to see if this is the very first chat room being made**

**if (strcmp(word,"Connected") != 0)**

**{**

**// Count the number of rooms based on the number of lines from bigbuf**

**int numRooms = 0;**

**for (int i=0;i<strlen(bigbuf)-1;i++)**

**{**

**numRooms += (bigbuf[i] == 9); // check for TAB characters**

**}**

**printf("Number of rooms: %d\n", numRooms);**

**// Make sure client's input is in the right format**

**while (1)**

**{**

**// Client specifies room number or "new"**

**memset(buffer, 0, 256);**

**fgets(buffer, 255, stdin);**

**int digitFlag = 1;**

**/\* Confirm whether every part of the message, besides \n, was an integer \*/**

**for (int i=0;i<strlen(buffer)-1;i++)**

**{**

**if (buffer[i] < 48 || buffer[i] > 57) // chars 0 to 9 are 48 to 57**

**{**

**digitFlag = 0;**

**break;**

**}**

**}**

**printf("digitFlag: %d\n", digitFlag);**

**// Did client enter a valid room number or "new"**

**if ((digitFlag && atoi(buffer) <= numRooms)|| strcmp(buffer,"new\n") == 0) break;**

**else printf("\nChoose a room number or type [new] to create a new room: ");**

**}**

**// Send new room number**

**// Demolish that newline character \n**

**if (buffer[strlen(buffer)-1] == 10) buffer[strlen(buffer)-1] = '\0';**

**int n = send(sockfd, buffer, strlen(buffer), 0);**

**if (n < 0) error("ERROR writing to socket");**

**}**

**}**

// start up threads

pthread\_t tid1;

pthread\_t tid2;

ThreadArgs\* args;

int errno = 0;

args = (ThreadArgs\*) malloc(sizeof(ThreadArgs));

args->clisockfd = sockfd;

errno = pthread\_create(&tid1, NULL, thread\_main\_send, (void\*) args);

// printf("errno1 %d\n", errno);

args = (ThreadArgs\*) malloc(sizeof(ThreadArgs));

args->clisockfd = sockfd;

errno = pthread\_create(&tid2, NULL, thread\_main\_recv, (void\*) args);

// printf("errno2 %d\n", errno);

// parent will wait for sender to finish (= user stop sending message and disconnect from server)

pthread\_join(tid1, NULL);

close(sockfd);

printf("\nFile descriptor closed connection\n");

return 0;

}

**Implementation**

Highlighted in **red** is the code that displays the up-to-date list of connected clients. Using the print\_client\_list() procedure, we can traverse the user linked list and print each client’s information with each connection of a client to the server.

Highlighted in **blue** shows the code necessary for accepting multiple main\_client connections, broadcasting messages from one client to all connected clients properly. When main\_server receives messages from a client, a broadcast() function is called. In this function, we send the client’s message, formatted, back out to every client except for the sender. This function came with the boiler plate code provided for the project, but it contained several bugs, the most profound being that the buffer arrays that held the messages needed to be cleared before storing and sending more data back out to the clients.

Highlighted in **orange** is the code that allows the client to specify username when connecting to the server. Upon establishing a connection to the main\_server, the main\_client asks the user to specify a ≤12 character username to be called by. Sent back to the main\_server, this username is stored in a user node struct, and thereby can be called later for other procedures such as displaying messages by that client to other clients, displaying the current connected client list, and disconnecting the client from the server.

Highlighted in **purple** is the chat room implementation for the network. The client specifies the chat room they want to join as an argument when calling main\_client. The client can either specify “new” if they want to create a new room, or a digit if that room already exists. By saving this argument to a buffer, we can send this buffer to the server and check to see what type of argument it is. If “new”, we create a new room for the client and set a new room number into the user struct. If a digit, we check to see if the room number exists on the server. If a valid digit, we place the client in the chat room, update their user struct properties, and send an acknowledgement back to the client saying that they joined the chat room successfully. If not, block the client from entering the server. The broadcast() function is edited to have a check to only send messages to clients with the same chat room number.

Highlighted in **dark red** shows the code for allowing the client to choose which chat room to join given no chat room argument when running main\_client. If the client specifies no argument, we let the server know this so we can present two options to them. First case: if no chat rooms have been created yet, just run the same output as if the “new” keyword was provided. If chat rooms exist, we send a large buffer of room choices back to the client to choose from, showing the room number and the amount of people in the room. The client will be stuck in an infinite loop until they enter one of two choices: the “new” keyword, or one of the currently existing rooms. From there, the client will be able to choose a username and get connected to their newly specified room number.

Highlighted in **green** gives a random unique color for each client in each room. The main\_client gets no choice here - once the client connects to a chat room, they’re assigned a random color using the choose\_color() procedure, and this attribute is kept in their user struct in the linked list. Every time they send a chat message, the client’s name will be in a random colored font.