Heavenly Medina and Joshua Grabenstein
CSC 345-01
Project 4 - report.pdf
May 24, 2019

Program Requirements (70 pts):
• (10 points) main_server can display the up-to-date list of connected clients. When a new client connects or any client disconnects, the up-to-date list will be printed in the server side terminal. It should work exactly in the following fashion: [in server terminal] 2 $ ./main_server [in client terminal] $ ./main_client IP-address-of-server

main_server.c

```
void clientList(){
	USR* temp = head;

	while(temp != NULL){
		printf("username: %s \n", temp->username);//Print statement
will change once we start adding ip addresses and usernames
		temp = temp->next;
	}
}
```

In main_server.c the updated client list is continually printed whenever someone either leaves or enters the server. It initializes a temp USR srtuct and iterates through the sll of structs until it is NULL to print out an updated list of users based on usernames. This function is called within the clientRemove(int ...) and add_tail(int ...;, char*...) methods within main_server.c.

• (10 points) main_server accepts multiple main_client connections and broadcast messages from one client to all connected clients properly.
This is done using the send() and recv() functions in main_server.c through the methods broadcast(...) and send messages within the function thread_main_send(..) and receives them through thread_main_recv(...) and buffer techniques.

main_client.c

```
void* thread_main_recv(void* args)
{
	pthread_detach(pthread_self());

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

	// keep receiving and displaying message from server
	char* buffer = (char*) malloc(512);
	int n;
```

```
	n = recv(sockfd, buffer, 512, 0);
	while (n > 0) {
		if (n < 0) error("ERROR recv() failed");

		setRandCol();

		printf("\n%s\n", buffer);
		buffer = (char*) malloc(255);
		n = recv(sockfd, buffer, 256, 0);

		printf(RESET);
	}

	return NULL;
}

void* thread_main_send(void* args)
{
	pthread_detach(pthread_self());

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

	// keep sending messages to the server
	char* buffer = (char*) malloc(255);
	int n;

	while (1) {
		// You will need a bit of control on your terminal
		// console or GUI to have a nice input window.
		printf("\n---------------------------------------------\n");
		printf("Please enter the message: ");
		buffer = (char*) malloc(255);
		fgets(buffer, 255, stdin);

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

		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
	}
	return NULL;
}
```

• (10 points) main_client can specify a username when connecting to the server. The server will maintain the user names and broadcast the names to clients properly.
The username is taken in through a buffer in main_client.c called usrnmBuff. The username is then passed and stored in the USR struct in main_server.c in a char pointer called username, and it is updated when a tail is added to the USR linked list of structs.

main_client.c

```
printf("Enter a username: \n");
      char* usrnmBuff = (char*) malloc(255);
      fgets(usrnmBuff, 255, stdin);
      n = send(sockfd, usrnmBuff, strlen(usrnmBuff), 0);
```

main_server.c

```
main(...){...
char* buffer = (char*) malloc(255);//Code for adding username!
            n = recv(newsockfd, buffer, 255,0);
            for(int i = 0; i < 256; i++){
            if(buffer[i] == '\n'){
                  buffer[i] = '\0';
                  //clientRemove()??
                  break;
                  }
            }
```

• (15 points) main_server allows multiple chatting rooms running simultaneously. For this purpose, the client behavior will be slightly changed. Client can request whether it will open a new chatting room or join an existing room. If a client wants to open a new chat room, then it should connect to the server as [in client terminal] $ ./main_client IP-address-of-server new If the connection was successful, then the client will be assigned a room number from the server and display it on the screen, e.g. [in client terminal] $ ./main_client IP-address-of-server new Connected to IP-address-of-server with new room number XXX If the client wants to connect to an existing room, then it can use a command as [in client terminal] $ ./main_client IP-address-of-server XXX You can assume that all rooms are numbered. main_server is responsible for maintaining the rooms and clients therein properly. If a main_client requests for non-existing room number, it should properly block the request. You can optionally set the maximum number of rooms to be managed by the server, but it should be at least 3.

We created two methods to create a new chat room or to join an existing chat room given a valid number, otherwise it automatically will join a random, valid chat room. There methods are called within the main method where there is a series of else if statements to determine what exactly the user has entered in the command line. If the user has entered

main_client.c

```
int main(...){
int* buffer = (int*) malloc(256);
      int n;
```

```
	n = recv(sockfd, buffer, 256*sizeof(int), 0);
	howManyRooms = buffer[0];
	newRoom(sockfd);
	printf("After\n");*/

	if(argc == 3){
	//printf("Check 1\n");
	int* buffer = (int*) malloc(256);
	int n;
	n = recv(sockfd, buffer, 256*sizeof(int), 0);
	howManyRooms = buffer[0];
	if(strcmp(argv[2], news) == 0){		//create a new room
			//receiverm(sockfd);
		newRoom(sockfd);
		}else if(argv[2] != NULL) {
			int passnum = atoi(argv[2]);
			//printf("From command line: %d\n", passnum);
			chooseRoom(passnum,sockfd);
			}
		}

void newRoom(int sockfd){
	howManyRooms++;
	  //add_room_tail(howManyRooms, 1);
	passnum(howManyRooms,sockfd);
}

void chooseRoom(int passNum, int sockfd){
	int n;
	int chatRoomNum = passNum;
	if(checkValidRoom(chatRoomNum) == true){	//join existing room
--> need to make sure room exists and that it's open
	  passnum(chatRoomNum,sockfd);
	/*char* buff = (char*) malloc(255);
	sprintf(buff, "%d\n",chatRoomNum);
	n = send(sockfd, buff, 255, 0);*/
	}else{
		printf("This input is invalid. Generating random room...");
		chatRoomNum = randRoom(chatRoomNum);
		passnum(chatRoomNum,sockfd);
		/*char* buff = (char*) malloc(255);
		sprintf(buff, "%d\n",chatRoomNum);
		n = send(sockfd, buff, 255, 0);*/
	}
}
```

• (10 points) Allow random joining of the existing room if the client requested [in client terminal] $ ./main_client IP-address-of-server assuming the feature above was implemented. If there is no room, then a new one should be created and assigned to this client.

main_client.c

```
int randRoom(int chatRoomNum){
	int ran; //random number, seed set within main function
	if(howManyRooms == 0){
		ran = 0;
	}else{
		ran = rand() % howManyRooms + 1;
	}
	if(checkValidRoom(ran) == false){//If there are no rooms, ran will
be zero, making the check return false, so we create a new room.
		howManyRooms++;
			chatRoomNum = howManyRooms;
	}else{
			chatRoomNum = ran;
		}
	return chatRoomNum;
}
```

Random joining is available through the randRoom function which generates a random number based on how many rooms are available using the seed determined in the main method. If there are no rooms available then 0 rooms are generated and the program will automatically generate a new room for the user to join. Otherwise, the method will allow for users to enter a randomly generated room number if there are any available. This method can be called by entering any value other than the correct number of rooms available or any string other than “new”.

• (15 points) Instead of explicitly specifying or random joining the room, let the client retrieve the list of the rooms available currently and choose from the list as below: [in client terminal] $ ./main_client IP-address-of-server Server says following options are available: Room 1: 2 people Room 2: 1 person … Room 10: 5 people Choose the room number or type [new] to create a new room: _ 3 If no room is available at the moment, it should automatically join to the new room (thus implicitly using new command above).

This portion determines on gathering input regarding how many rooms and how many numbers of users there are through the main_server.c, which it is sent through the send() and recv() functions using buffers. Once the information regarding each room is successfully transmitted then it goes through the main method within main_server.c to determine what exactly the user requested to do, whether it be generate a new room or join an existing room using the newRoom() and chooseRoom() methods as shown above for

the previous requirement. This is done using an else if series of statements once the progra has determined that the command was entered after the main client has joined using the ip address but has not yet been assigned to a room.

```
else if(argc != 3){
                  //printf("Check 2\n");
                  if(printBuff(sockfd) == false){
                        newRoom(sockfd);
                        }else{
                              char* passVal = (char*)malloc(128);
                              fgets(passVal,128,stdin);
                              passVal[strcspn(passVal, "\n")] = '\0';
                              if(strcmp(passVal,"new") == 0){
                                    newRoom(sockfd);
                              }else{
                                    int passNum;
                                    passNum = atoi(&passVal[0]);
                                    chooseRoom(passNum,sockfd);
                                    }
                        }
            }else{                                    //join a random room
if available
            chatRoomNum = randRoom(chatRoomNum);
            passnum(chatRoomNum,sockfd);
             }
...}
```

Extra Credit:

• (10 points) Use a random, unique color for each client in a room. It is okay if clients are using different sets of colors for the same room information, as long as individual users are distinguishable with different colors.

define different colors to use in the terminal at the start of the main_client.c program

```
//used for colors
#define RED "\x1B[31m"
#define GREEN "\x1B[32m"
#define MAG "\x1B[35m"
#define BLUE "\x1B[34m"
#define CYN "\x1B[36m"
#define RESET "\x1B[0m"
```

create a function to set the color randomly by generating a random number using time.h

main_client.c

```
int main(...){
...
srand((unsigned)time (NULL));
...}

void setRandCol(){
      int ran = rand() % 5 + 1;

      if(ran == 1){
            printf(RED);
      }else if(ran == 2){
            printf(GREEN);
      }else if(ran == 3){
            printf(MAG);
      }else if(ran == 4){
            printf(BLUE);
      }else{
            printf(CYN);
      }
}

void* thread_main_recv(void* args){
...
setRandCol();

            printf("\n%s\n", buffer);
            buffer = (char*) malloc(255);
            n = recv(sockfd, buffer, 256, 0);

printf(RESET);
...}
```

The color is randomly set among different users in chat rooms. The colors are initially defined at the start of the program and are randomly selected by generating a random number from 1-5 (5 being the number of colors available/defined to use) and picking a random color according to the number generated through a series of else if statements using printf(random color). It later sets the color within the function thread_main_recv(...) when setRandCol() is called, and the color is later reset when printf(RESET) is called within the same method. This results in a random color for each user who enters a chatroom in the client terminal.