Myles Greene-Beaupre 05/11/2022
C12195336
ECE 470

Multiplayer Trivia Game
Project 2 Final Report

Project Description

This final project for ECE 470 is the implementation of a multi-user trivia game which allows for an indefinite amount of players to play against each other in a designated number of rounds of multiple choice trivia questions. It is implemented using the C programming language and UDP unix sockets, and the server runs on an Arch Linux distribution running using the Windows Subsystem for Linux (WSL). There are 22 categories for trivia questions, and each category has about 8000 lines of questions, so it would take a long time before a player could go through every question. All of the code is custom, and the project does not depend on anything external except the files with the questions and basic unix sockets/networking capabilities.

General Architecture

The architecture of the server side involves a while loop that listens for UDP packets and processes them according to the first character of the message, which determines the intention of the packet, for example a packet with the first character J indicates that a user wants to join the game. It receives a packet using the receive_message() function, which simply calls recvfrom() and passes it a struct sockadder* so that the function can fill it with information about the person who sent the packet (this is necessary for the server to send packets back to the proper client). Sending messages is handled by the send_message() function, which calls sendto() with error checking, and initializing a socket is done by the init_socket() function which calls socket() and fills in information for the struct sockaddr_in* it was passed. These three functions (init_socket(), send_message(), and receive_message()) are the basis for setting up the networking and sending/receiving messages, and they are found in common.c which is used by both server.c and client.c. When the server gets a packet, it calls the appropriate server function such as add_player() or set_ready(). The actual game starts after a check in set_ready(), which determines if more than one client and all connected clients are readied up, in which it calls start_game() and goes through the game loop. The client has much less functionality, and pretty much just prints messages it receives from the server. The whole program is mostly single-threaded except for the case in which user input on the client side must be interrupted (this happens if the user has not entered a response to a question and 15 seconds have passed), and also when the server side is going through the game loop and has to sleep for 15 seconds, it uses another thread to receive packets while the main thread is sleeping. The server keeps track of player information through a struct player, and there is an array of this player struct contained inside the game struct, which also holds game state information. This game struct is what is passed around to all of the necessary functions, so that they can access information about the game as well as information about every player in the game (most importantly their address so they can be sent messages). This is the basic architecture, overall it is very server heavy and the client pretty much takes user input and sends it to the server, and waits for responses to be printed. The files consist of client.c (main), server.c (main), clientfuncs.c, clientfuncs.h, serverfuncs.c, serverfuncs.h, common.c, common.h, and all the files which hold trivia questions (there are 22 of these, giving us 22 categories and the server chooses a random category every game).

Scenario 1: Join Game (client → server)

The client sends a join game message containing their username to the server, assuming that the server has been started. If the server determines that a game is in progress, the client will be put on a waitlist until the game ends. Once the game ends, the client will be put into the pre-game lobby for a new game and will be sent a message telling them to enter ‘R’ when they are ready. If a game was not already in progress, they will be put into this pre-game lobby immediately instead. In both cases, if the user enters a name that is already in use, either in a game currently being played, on the waitlist, or in the pre-game lobby, they will be prompted by the server for a different name and will not be put into the pre-game lobby or waitlist until they enter an unused name.

Scenario 2: Player Ready (client → server)

Once a client has done the join game scenario, the server will send them a message telling them to send ‘R’ when they are ready. A message containing anything else will be ignored by the server, and the client will be informed of this. The ready message will also contain the user’s name, so when the server gets an ‘R’ from a user it will send to all users in the lobby a list of everyone they are playing with, and whether or not they are ready to play (so everyone in the lobby will be able to see who is delaying the game).

Scenario 3: Answer Question (client → server)

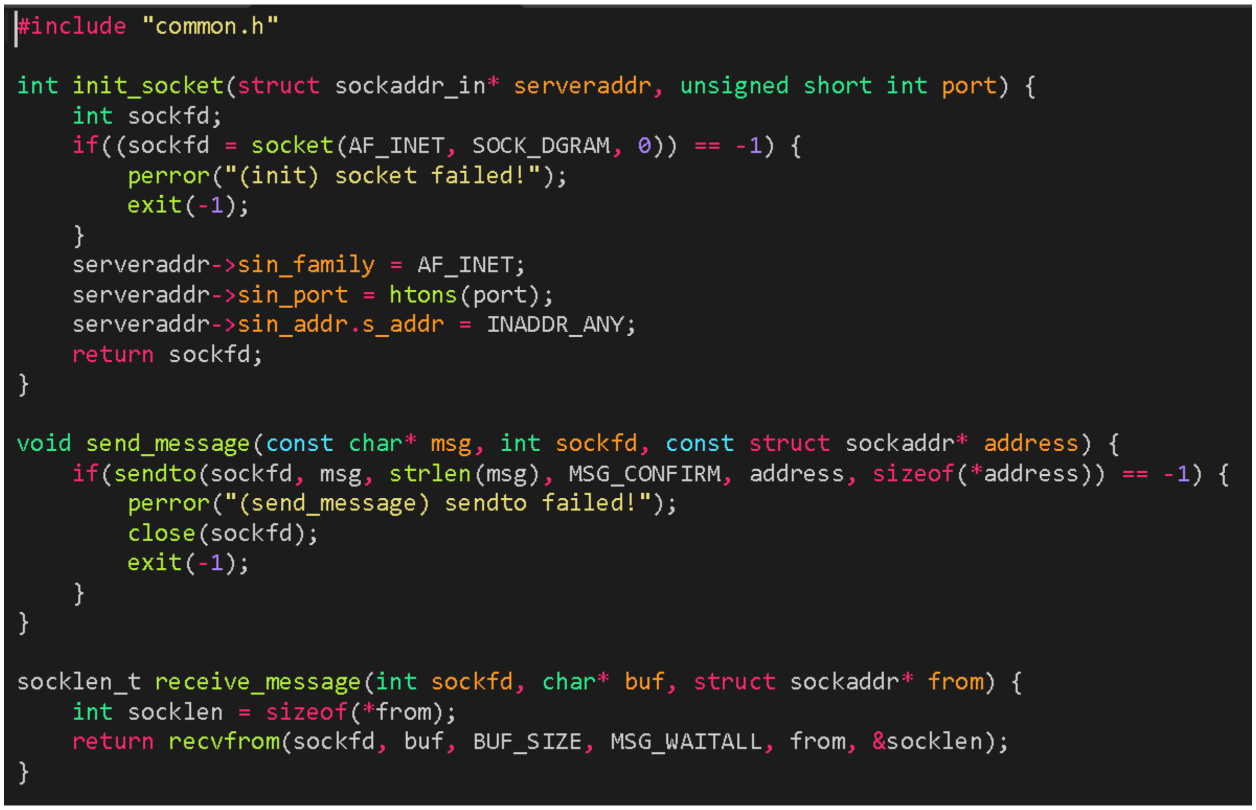
Answer messages will be ignored by the server unless a game is in progress, and the username contained in the anwer message is the name of a user currently playing in the game (not waitlisted). When a server receives an answer message, it will check to see if the answer contained within it matches the answer to the last trivia question that was displayed to the players. When it is checking to see if a player’s answer matches the correct answer, capitalization will be ignored but if you spell the answer incorrectly and/or your answer does not match the length of the correct answer, it will be considered incorrect. If the server determines an answer is wrong, the person who sent the message will get a message back indicating it was incorrect, and they will be allowed to enter another answer. If it was correct, the server will calculate the number of points to award based on how many people got the answer right before them, it will increment their points on the server side, and then send a message to the user that submitted the answer indicating that it was correct and how many points they earned that round, as well as their total number of points for the whole game.

Scenario 4: Global Message (server → all clients)

This scenario is more general, but it is any situation in which all of the players in the game must be given the same information to display on their screens. The server will construct the message, and then send it to all clients with an indication that it should be printed to the player on the client side. This scenario will happen in the pre-game lobby, whenever a client indicates that they are ready every user that is currently in the lobby will be sent an updated list of everyone who is ready and everyone who is not. This scenario will also happen for displaying the same trivia question to all players of the game, and for messages such as "a game is starting soon." Finally, this scenario will happen at the end of the final round in a game, in which case every player will receive an ordered list of all players and their score with the winner at the top, then second place, then third, etc. all the way to last place. By abstracting all of these situations to one scenario, the system is made much simpler since this "global message" scenario will be implemented as a function that can be used in many cases.

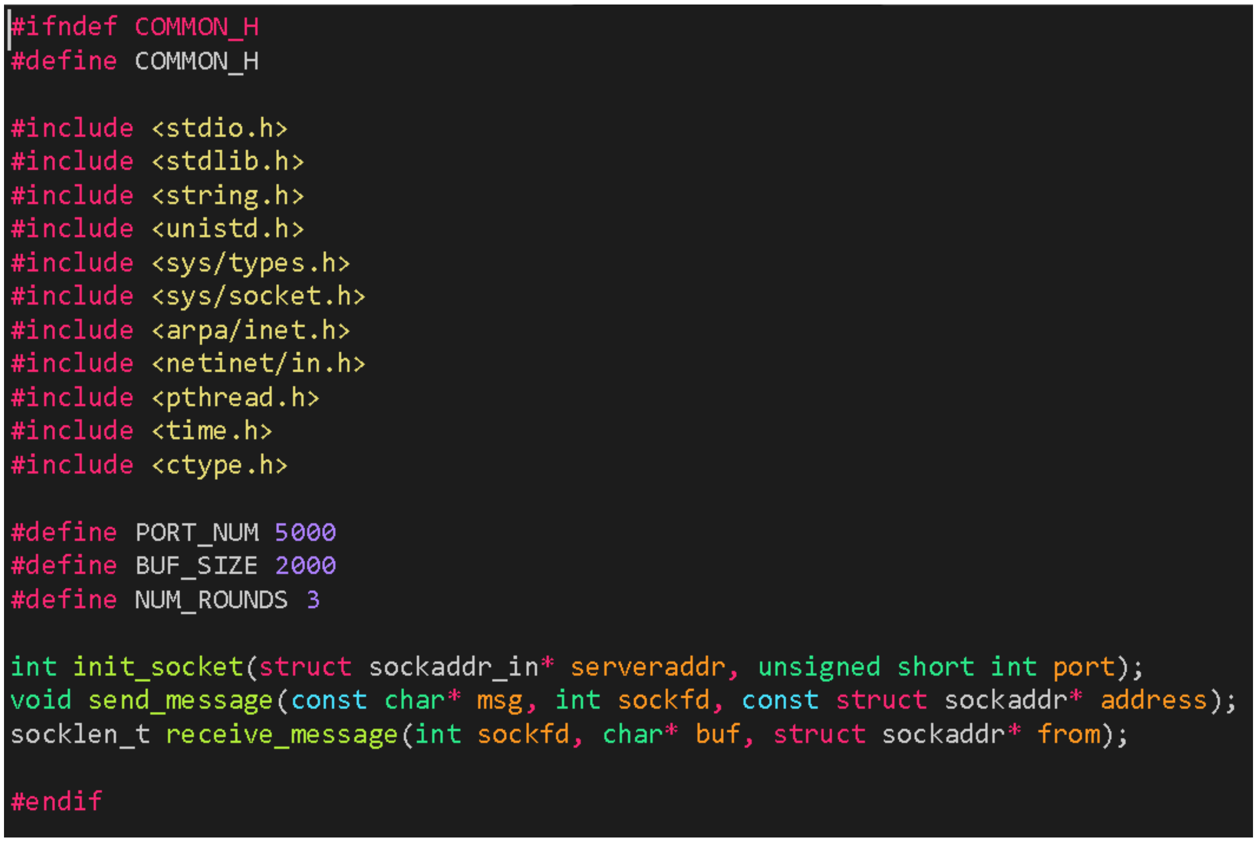
Detailed Design

common.c



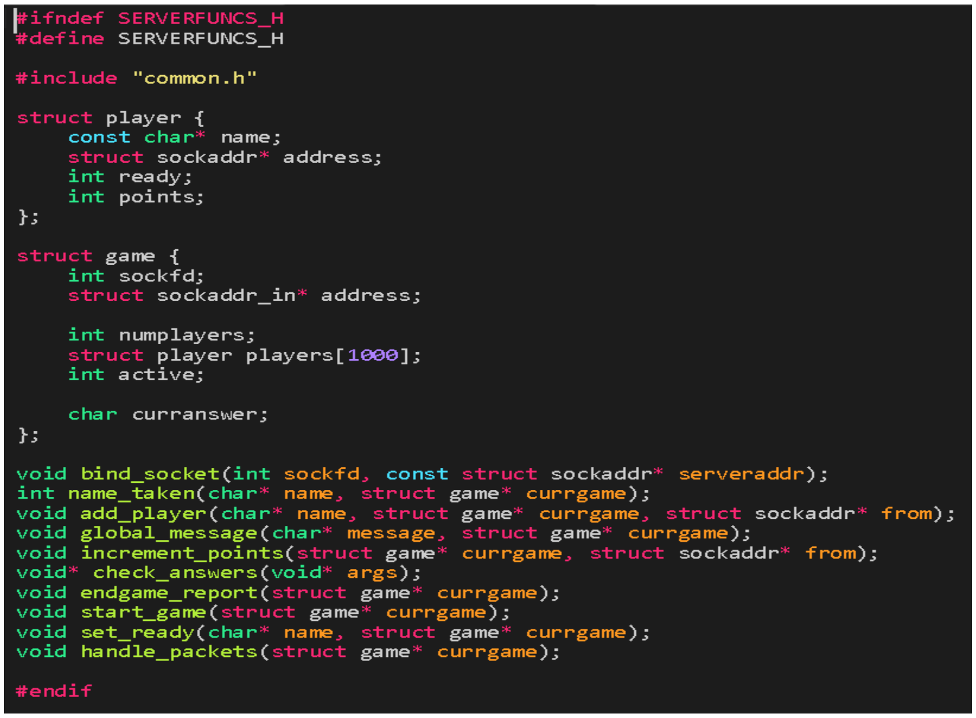
```c
#include "common.h"

int init_socket(struct sockaddr_in* serveraddr, unsigned short int port) {
    int sockfd;
    if((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
        perror("(init) socket failed!");
        exit(-1);
    }
    serveraddr->sin_family = AF_INET;
    serveraddr->sin_port = htons(port);
    serveraddr->sin_addr.s_addr = INADDR_ANY;
    return sockfd;
}

void send_message(const char* msg, int sockfd, const struct sockaddr* address) {
    if(sendto(sockfd, msg, strlen(msg), MSG_CONFIRM, address, sizeof(*address)) == -1) {
        perror("(send_message) sendto failed!");
        close(sockfd);
        exit(-1);
    }
}

socklen_t receive_message(int sockfd, char* buf, struct sockaddr* from) {
    int socklen = sizeof(*from);
    return recvfrom(sockfd, buf, BUF_SIZE, MSG_WAITALL, from, &socklen);
}
```

common.h



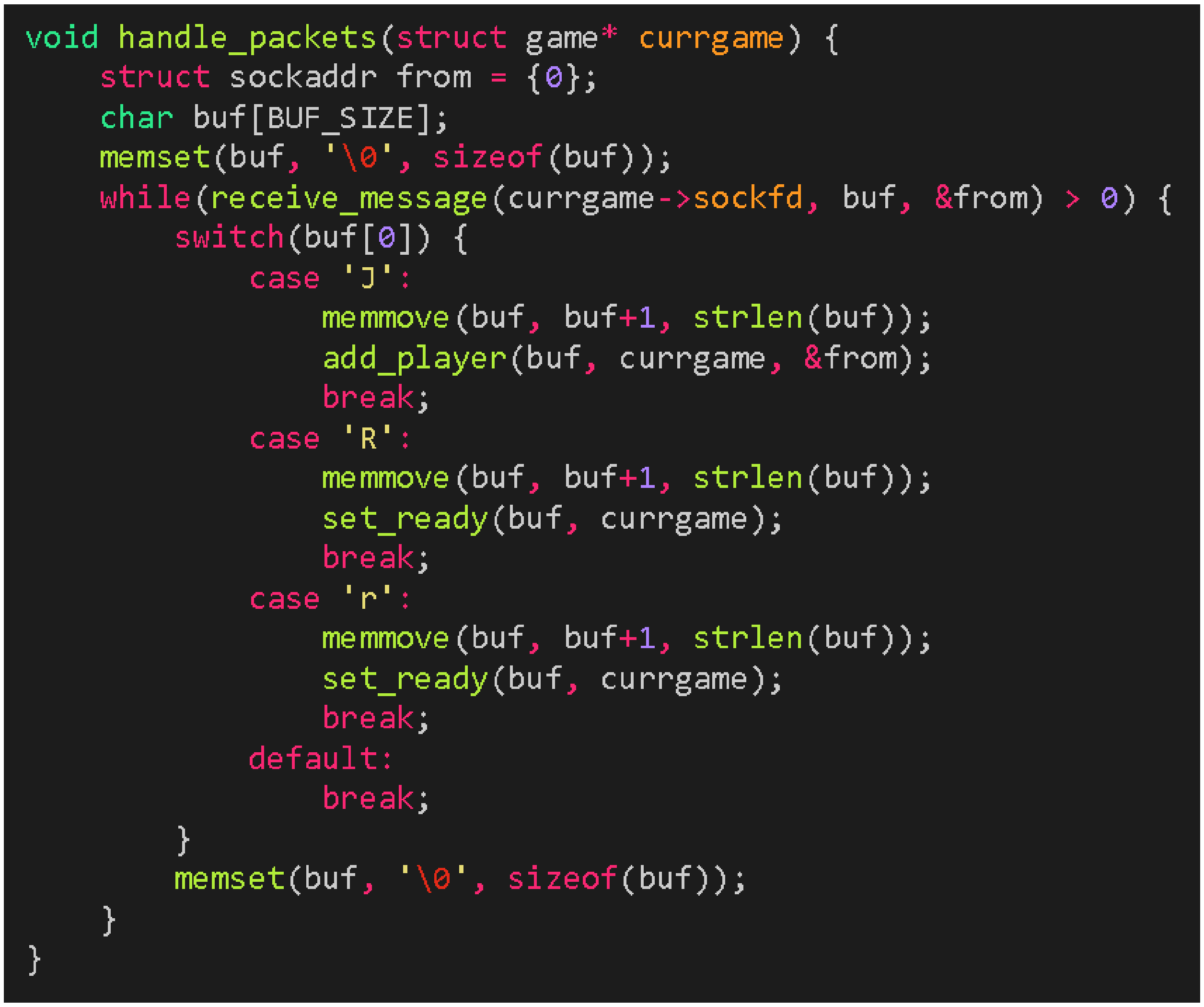
```c
#ifndef COMMON_H
#define COMMON_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <pthread.h>
#include <time.h>
#include <ctype.h>

#define PORT_NUM 5000
#define BUF_SIZE 2000
#define NUM_ROUNDS 3

int init_socket(struct sockaddr_in* serveraddr, unsigned short int port);
void send_message(const char* msg, int sockfd, const struct sockaddr* address);
socklen_t receive_message(int sockfd, char* buf, struct sockaddr* from);

#endif
```

serverfuncs.h



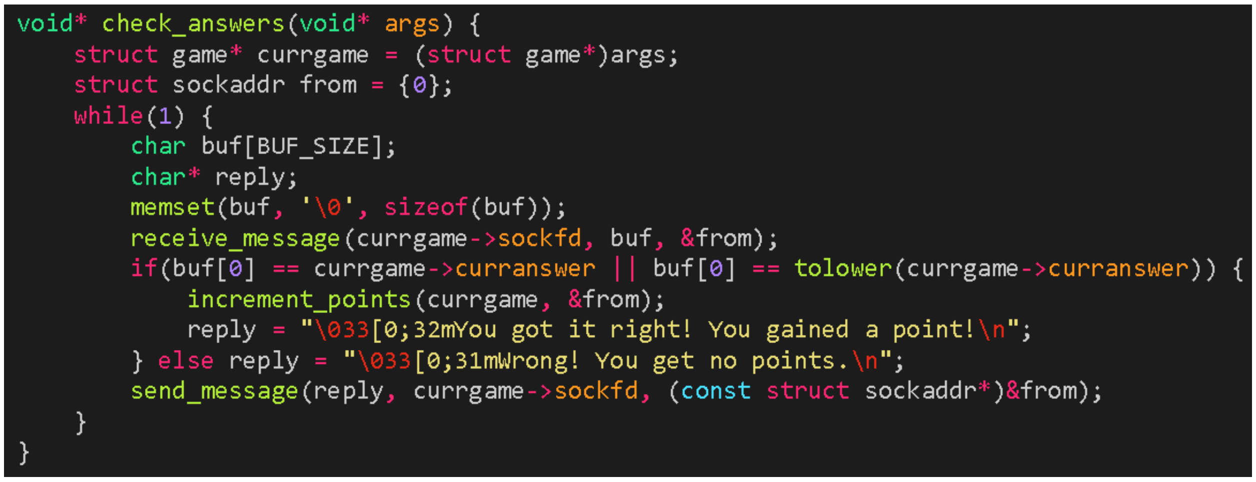
```c
#ifndef SERVERFUNCS_H
#define SERVERFUNCS_H

#include "common.h"

struct player {
    const char* name;
    struct sockaddr* address;
    int ready;
    int points;
};

struct game {
    int sockfd;
    struct sockaddr_in* address;

    int numplayers;
    struct player players[1000];
    int active;

    char curranswer;
};

void bind_socket(int sockfd, const struct sockaddr* serveraddr);
int name_taken(char* name, struct game* currgame);
void add_player(char* name, struct game* currgame, struct sockaddr* from);
void global_message(char* message, struct game* currgame);
void increment_points(struct game* currgame, struct sockaddr* from);
void* check_answers(void* args);
void endgame_report(struct game* currgame);
void start_game(struct game* currgame);
void set_ready(char* name, struct game* currgame);
void handle_packets(struct game* currgame);

#endif
```

how join and ready packets are handled (pre-game)



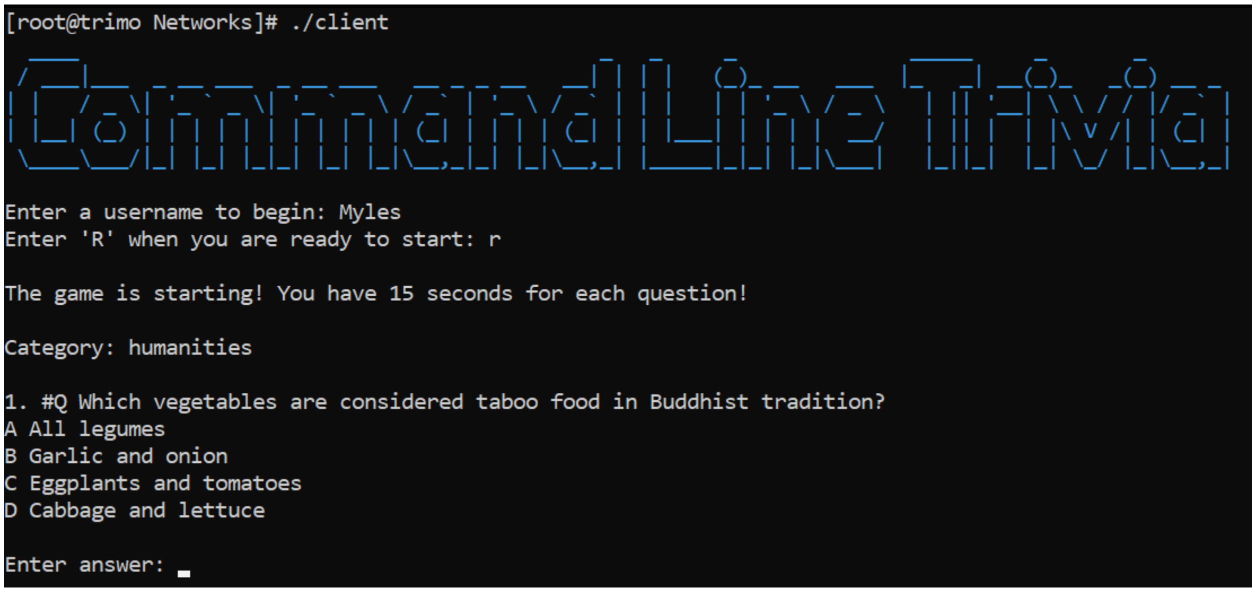
```c
void handle_packets(struct game* currgame) {
    struct sockaddr from = {0};
    char buf[BUF_SIZE];
    memset(buf, '\0', sizeof(buf));
    while(receive_message(currgame->sockfd, buf, &from) > 0) {
        switch(buf[0]) {
            case 'J':
                memmove(buf, buf+1, strlen(buf));
                add_player(buf, currgame, &from);
                break;
            case 'R':
                memmove(buf, buf+1, strlen(buf));
                set_ready(buf, currgame);
                break;
            case 'r':
                memmove(buf, buf+1, strlen(buf));
                set_ready(buf, currgame);
                break;
            default:
                break;
        }
        memset(buf, '\0', sizeof(buf));
    }
}
```

how answer packets are handled

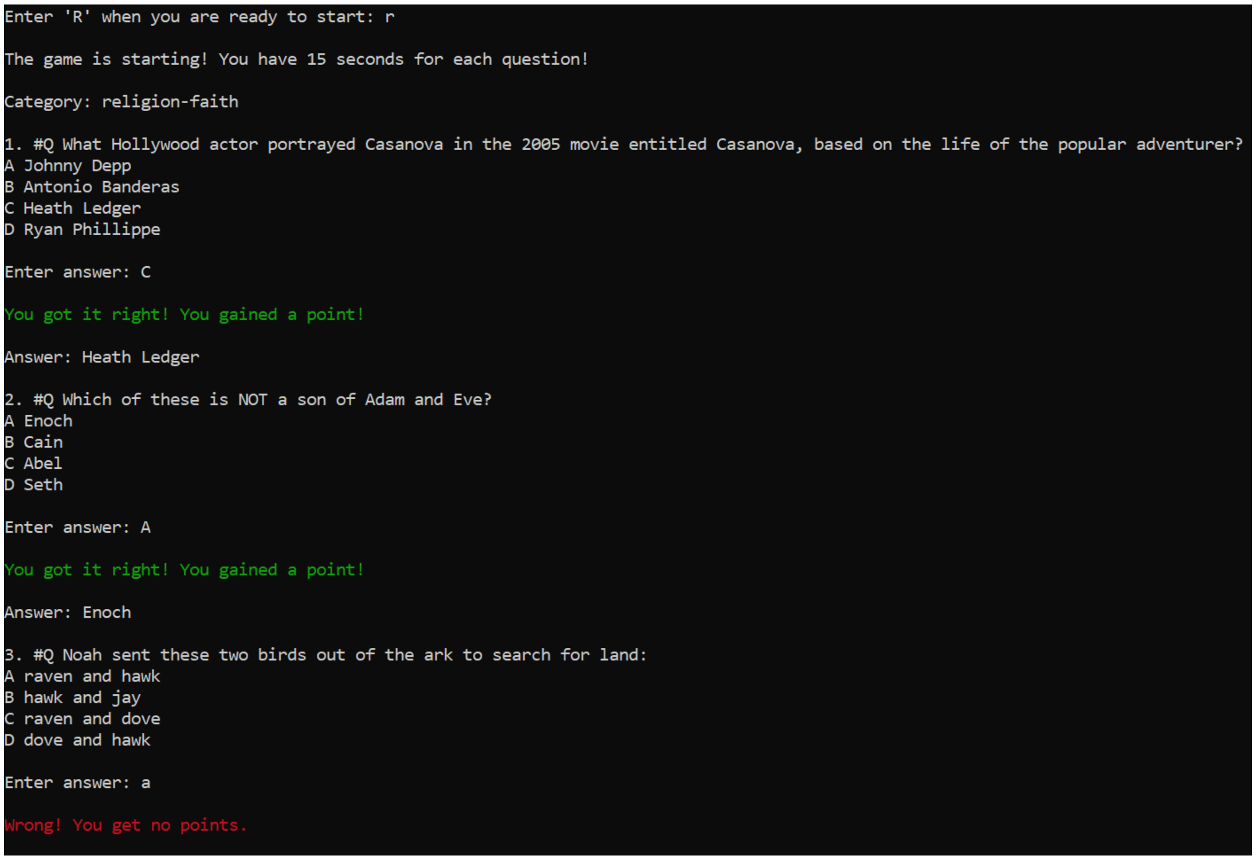


```c
void* check_answers(void* args) {
    struct game* currgame = (struct game*)args;
    struct sockaddr from = {0};
    while(1) {
        char buf[BUF_SIZE];
        char* reply;
        memset(buf, '\0', sizeof(buf));
        receive_message(currgame->sockfd, buf, &from);
        if(buf[0] == currgame->curranswer || buf[0] == tolower(currgame->curranswer)) {
            increment_points(currgame, &from);
            reply = "\033[0;32mYou got it right! You gained a point!\n";
        } else reply = "\033[0;31mWrong! You get no points.\n";
        send_message(reply, currgame->sockfd, (const struct sockaddr*)&from);
    }
}
```

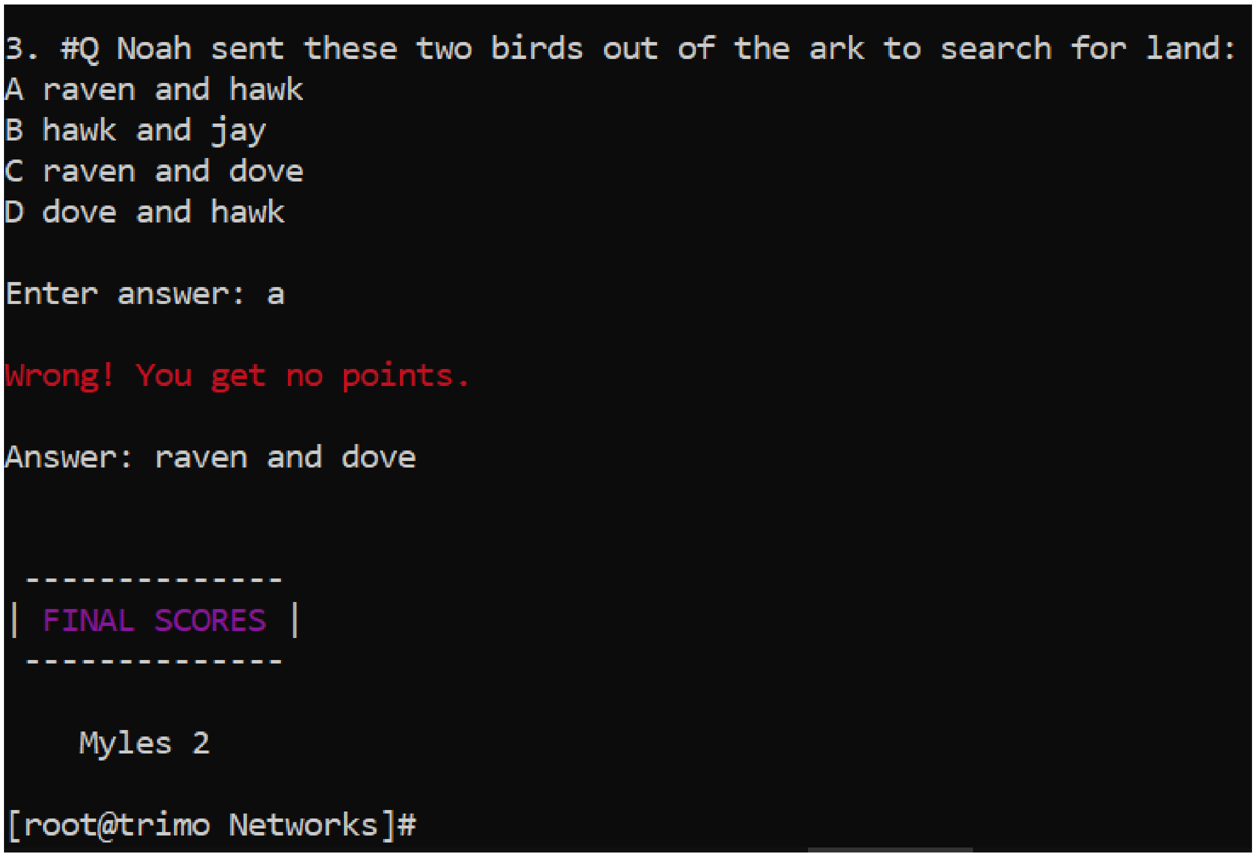
Program Demo Screenshots



```
[root@trimo Networks]# ./client
  ____                                          _   _     _              _____     _       _
 / ___|___  _ __ ___  _ __ ___   __ _ _ __   __| | | |   (_)_ __   ___  |_   _| __(_)_   _(_) __ _
| |   / _ \| '_ ` _ \| '_ ` _ \ / _` | '_ \ / _` | | |   | | '_ \ / _ \   | || '__| \ \ / / |/ _` |
| |__| (_) | | | | | | | | | | | (_| | | | | (_| | | |___| | | | |  __/   | || |  | |\ V /| | (_| |
 \____\___/|_| |_| |_|_| |_| |_|\__,_|_| |_|\__,_| |_____|_|_| |_|\___|   |_||_|  |_| \_/ |_|\__,_|

Enter a username to begin: Myles
Enter 'R' when you are ready to start: r

The game is starting! You have 15 seconds for each question!

Category: humanities

1. #Q Which vegetables are considered taboo food in Buddhist tradition?
A All legumes
B Garlic and onion
C Eggplants and tomatoes
D Cabbage and lettuce

Enter answer:
```



```
Enter 'R' when you are ready to start: r

The game is starting! You have 15 seconds for each question!

Category: religion-faith

1. #Q What Hollywood actor portrayed Casanova in the 2005 movie entitled Casanova, based on the life of the popular adventurer?
A Johnny Depp
B Antonio Banderas
C Heath Ledger
D Ryan Phillippe

Enter answer: C

You got it right! You gained a point!

Answer: Heath Ledger

2. #Q Which of these is NOT a son of Adam and Eve?
A Enoch
B Cain
C Abel
D Seth

Enter answer: A

You got it right! You gained a point!

Answer: Enoch

3. #Q Noah sent these two birds out of the ark to search for land:
A raven and hawk
B hawk and jay
C raven and dove
D dove and hawk

Enter answer: a

Wrong! You get no points.
```



```
3. #Q Noah sent these two birds out of the ark to search for land:
A raven and hawk
B hawk and jay
C raven and dove
D dove and hawk

Enter answer: a

Wrong! You get no points.

Answer: raven and dove


 --------------
| FINAL SCORES |
 --------------

    Myles 2

[root@trimo Networks]#
```