Цель работы

1. Целью является закрепление и использование знаний и практических навыков, приобретенных в течении курса.

2. Исследоваие предложенной предметной области и реализация конкрентного, работоспособного программного прототипа

Задание

Реализовать многопользовательскую клиент-серверную игру «Морской бой», работающую в консоли.

Требования к программе:

1. Поддержка нескольких игр сразу (активные игровые сессии и ожидание оппонента)

2. Обработка досрочного завершения игры (выход одного из соперников из сессии)

3. Рейтинг всех игроков, запускавших игру с возможностью подсчета их "win rate"

4. Технология взаимодействия пользователей на усмотрение (очереди сообщений, сокеты, пайпы и т.д.)

Необходимо обрабатывать системные ошибки, которые могут возникнуть в результате работы.

Описание программы

Так как игра клиент-серверная, то по сути проект состоит из двух разных программ. Одна программа – это сервер, другая, соответственно – клиент. Экземпляры программ-клиентов могут взаимодействовать с программами-серверами при помощи обмена сообщениями с помощью сокетов.

Файл `interact.h` содержит логику , формирование сообщений, их отправка и получение, а так же структуру сообщений, которая используется уже непосредственно для клиент-серверного взаимодействия.

Файл `database.h` содержит все, что касается импровизированной базы данных для подсчета рейтинга: чтение/запись, добавление, поиск записей.

Файл `field.h` содержит основную логику для управления полем и алгоритм случайной генерации игрового поля.

Файл `render.h` содержит логику для отрисовки игровых полей и управления цветом.

Файл `server.c` содержит, очевидно, логику работы серверного приложения.

Сервер работает следующим образом. Сервер загружает импровизированную базу данных из файла. Далее создается очередь – своеобразная комната ожидания, в которой вновь подключившиеся игроки будут дожидаться своего оппонента, с прибытием

которгого начнется игровая сессия. Далее создается пул игровых сессий, куда будут добавлять активные игровые сессии для удобства организации общения между клиентами и серверами и, как следствие между противниками через сервер. Далее создается мастер-сокет и привязывается к указанному порту. Он будет находится в режиме прослушивания порта на предмет подключения новых клиентов.

Далее инициализацию можно считать законченной, и серевер входит в основной цикл, выполняющийся до тех пор, пока его не попросят завершится. В основном цикле и происходит вся обработка как пользовательских запросов, так и прием новых подключений.

Для этого используется один из стандартных подходов: комбинация `fd_set` и `select`. В каждой итерации основного цикла множество заполняется интересующми нас сокетами. Далее к нему применяется процедура `select`. После ее применения в множестве остаются те дескрипторы, которым "есть что сказать"т.е. если они ожидают чтения. Далее просто считывается сообщение-запрос, и в зависимости от его типа формируется ответ и (если это необходимо) послыка другому клиенту, в данном случае оппоненту. Далее все повторяется.

В программе-сервере так же определены пользовательские обработчики системных сигналов `SIGINT` и `SIGTSTP` для остановки сервера и временной приостановки соответственно.

В конце работы сервера, когда произошло прерывание по сигналу `SIGINT`, сервер сохраняет данные в базу данных и освобождает ресурсы, закрывает сокеты.

Файл `client.c` содержит логику работы клиентского приложения.

В клиентском приложении происходит следующее. Создается сокет, производится попытка подключения к серверу. Далее инициализир экранный буффер. Позже программа входит в основной цикл работы. Пользователю предоставляется на выбор набор действий (главное меню). Здесь он может начать игру, запросить рейтинг или же выйти из программы. После того как игрок выбрал «play» на сервер отсылается сообщение означающее просьбу добавить игрока в очередь. Далее пользователь ждет прибытия своего оппонента. Потом сервер генерирует и рассылает обоим противникам игровые поля, сообщает кто первый ходит, и игра начинается. Игрок, у которого правхо ходить, стреляет пока не промахнется, либо же не выбъет все вражеские корабли тем самым одержав победу. Во время каждого выстрела происходит запрос с координатами выстрела и ожидание ответа от сервера, свидетельствующего о попадании или промахе. После каждого попадания на сервере происходит проверка есть ли «живые» корабли у обстреливаемого: если нет, то сервер завершает игровую сессию, поздравляет победителя, утешает проигравшего, заносит сведения о схватке в рейтинг и отправляет обоих игроков в главное меню. Потом все повторяется.

Когда пользователь изволит закончить работу вводом `exit`, программа выходит из бесконечного цикла, освобождает ресурсы и завершается.

Листинг

`interact.h`

```c
#ifndef INTERACT_H
#define INTERACT_H

#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include <errno.h>

#define MT_EMPTY 0
#define MT_LOGIN 1
#define MT_SHOT 2
#define MT_FIELD 3
#define MT_TURN 4
#define MT_HIT 5
#define MT_MISS 6
#define MT_WIN 7
#define MT_LOSE 8
#define MT_RATING 9

#define SR_MISS 0
#define SR_HIT 1

typedef struct{
    int type;
    int id;
    int size;
    void *buffer;
}msg_t;


typedef struct{
    int x;
    int y;
}shot;

void init_msg(msg_t *m){
    m->type = MT_EMPTY;
    m->size = 0;
    m->buffer = NULL;
}

void set_type_msg(msg_t *m, int type){
    m->type = type;
}

void set_id_msg(msg_t *m, int id){
    m->id = id;
}

void init_size_msg(msg_t *m, int size){
    m->buffer = malloc(size);
    m->size = size;
}

void close_msg(msg_t  *m){
    if(m->buffer != NULL)
        free(m->buffer);
    m->size = 0;
}

void init_data_msg(msg_t *m, void *data, int size, int type, int id){
```

```
    init_size_msg(m, size);
    memcpy(m->buffer, data, size);
    set_type_msg(m, type);
    set_id_msg(m, id);
}

int send_msg(int socket, msg_t *m){
    send(socket, &m->size, sizeof(int), 0);
    send(socket, &m->type, sizeof(int), 0);
    send(socket, &m->id, sizeof(int), 0);
    send(socket, m->buffer, m->size, 0);

    return 1;
}

int recieve_msg(int socket, msg_t *m){
    if(recv(socket, &m->size, sizeof(int), 0) == 0)
        return 0;
    init_size_msg(m, m->size);
    recv(socket, &m->type, sizeof(int), 0);
    recv(socket, &m->id, sizeof(int), 0);
    recv(socket, m->buffer, m->size, 0);

    return 1;
}

#endif
```

server.c

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include <signal.h>
#include <time.h>

#include "vector.h"
#include "database.h"
#include "interact.h"
#include "field.h"

#define MAX_CLIENT_C 256
#define READ_BUFF_L 256

#define MAX_SESSIONS_C 16

#define QUEUE_CAP 16

typedef struct{
    int sock;
    record *rec;
} player;

typedef struct{
    player player1;
    player player2;
    char field1[100];
    char field2[100];
    int turn;
} session;
```

```
int should_close = 0;

void sigint_handler(int signal){
    should_close = 1;
}

void end_session(session *s, int winner){
    assert(s != NULL);
    msg_t w_msg;
    init_data_msg(&w_msg, &w_msg, 1, MT_WIN, 0);

    if(winner == 0){
        send_msg(s->player1.sock, &w_msg);
        s->player1.rec->wins++;
        s->player2.rec->losses++;
    }
    else{
        send_msg(s->player2.sock, &w_msg);
        s->player2.rec->wins++;
        s->player1.rec->losses++;
    }
    close_msg(&w_msg);
}


int main(int argc, char **argv){
    char read_buff[READ_BUFF_L];

    if(argc < 3){
        printf("usage: server <port> <database_file>");
        exit(EXIT_SUCCESS);
    }

    signal(SIGINT, sigint_handler);

    database base = load(argv[2]);

    vector queue;
    vector_init(&queue, QUEUE_CAP, player);

    session *sessions[MAX_SESSIONS_C];
    for(int i = 0; i < MAX_SESSIONS_C; i++)
        sessions[i] = NULL;

    srand(time(NULL));
    rand();
    int port = atoi(argv[1]);
    printf("port: %d\n", port);

    int client_socks[MAX_CLIENT_C];
    for(int i = 0; i < MAX_CLIENT_C; i++)
        client_socks[i] = 0;

    int master_sock;
    if((master_sock = socket(AF_INET, SOCK_STREAM, 0)) == 0){
        perror("master socket failed\n");
        exit(EXIT_FAILURE);
    }

    int opt = 1;
    if(setsockopt(master_sock, SOL_SOCKET, SO_REUSEADDR
        | SO_REUSEPORT, &opt, sizeof(opt)) < 0){
        perror("set socket options failed\n");
        exit(EXIT_FAILURE);
    }
```

```
struct sockaddr_in addr;
addr.sin_family = AF_INET;
addr.sin_addr.s_addr = INADDR_ANY;
addr.sin_port = htons(port);

if(bind(master_sock, (struct sockaddr*)&addr, sizeof(struct sockaddr_in)) < 0){
    perror("master socket bind failed\n");
    exit(EXIT_FAILURE);
}

if(listen(master_sock, 3) < 0){
    perror("listen failed\n");
    exit(EXIT_FAILURE);
}

fd_set read_fd;

while (!should_close){
    FD_ZERO(&read_fd);
    FD_SET(master_sock, &read_fd);

    int max_fd = master_sock;

    for(int i = 0; i < MAX_CLIENT_C; i++)
        if(client_socks[i] > 0){
            FD_SET(client_socks[i], &read_fd);
            if(max_fd < client_socks[i])
                max_fd = client_socks[i];
        }

    if(select(max_fd + 1, &read_fd, NULL, NULL, NULL) < 0){
        if(errno != EINTR){
            perror("select failed\n");
            exit(EXIT_FAILURE);
        }
    }
    else{
        if (FD_ISSET(master_sock, &read_fd)){
            printf("new connection\n");
            int new_sock;
            if((new_sock = accept(master_sock, NULL, 0)) < 0){
                perror("accept failed\n");
                exit(EXIT_FAILURE);
            }
            printf("connected: %d\n", new_sock);
            for(int i = 0; i < MAX_CLIENT_C; i++){
                if(client_socks[i] == 0){
                    client_socks[i] = new_sock;
                    break;
                }
            }
        }
        for(int i = 0; i < MAX_CLIENT_C; i++){
            if(FD_ISSET(client_socks[i], &read_fd)){
                printf("processing client: %d\n", client_socks[i]);
                msg_t req;
                if(recieve_msg(client_socks[i], &req) == 0){
                    printf("disconnected\n");

                    int queue_size = vector_size(&queue);
                    for(int j = 0; j < queue_size; j++){
                        player temp;
                        vector_fetch(&queue, j, &temp);
                        if(temp.sock == client_socks[i]){
```

```
                                printf("dequeued disconnected player\n");
                                vector_erase(&queue, j);
                                break;
                            }
                        }
                        for(int j = 0; j < MAX_SESSIONS_C; j++){
                            int found = 0;
                            if(sessions[j] != NULL){
                                if(sessions[j]->player1.sock == client_socks[i]){
                                    sessions[j]->player1.rec->lefts++;
                                    end_session(sessions[j], 1);
                                    found = 1;
                                }
                                if(sessions[j]->player2.sock == client_socks[i]){
                                    sessions[j]->player2.rec->lefts++;
                                    end_session(sessions[j], 0);
                                    found = 1;
                                }
                                if(found){
                                    printf("session ended because of
____________________________________disconnected player\n");
                                    free(sessions[j]);
                                    sessions[j] = NULL;
                                    break;
                                }
                            }

                        }
                        printf("before closing\n");
                        close(client_socks[i]);
                        client_socks[i] = 0;
                    }
                    else{
                        switch (req.type){
                            case MT_LOGIN:{
                                    record *rec;
                                    printf("login %s\n", (char*)req.buffer);
                                    if((rec = find_rec(&base, req.buffer)) ==
                                    NULL){
                                        printf("player %s not found\n", (char*)req.buffer);
                                        record new_rec;
                                        strcpy(new_rec.uname, req.buffer);
                                        new_rec.wins = 0;
                                        new_rec.losses = 0;
                                        new_rec.lefts = 0;

                                        add(&base, new_rec);
                                        printf("added to database\n");
                                        rec = get_rec(&base, base.count - 1);
                                    }
                                    printf("statistics:\n");
                                    printf("\twins: %d\n", rec->wins);
                                    printf("\tlosses: %d\n", rec->losses);
                                    printf("\tlefts: %d\n", rec->lefts);

                                    player p = {client_socks[i], rec};
                                    vector_push_front(&queue, &p);
                                    printf("queued\n");

                                    if(vector_size(&queue) >= 2){
                                        printf("start new session\n");
                                        session *s;
                                        int i;
                                        for(i = 0; i < MAX_SESSIONS_C; i++){
                                            if(sessions[i] == NULL){
```

```
                    sessions[i] = malloc(sizeof(session));
                    s = sessions[i];
                    break;
                }
            }
            vector_back(&queue, &s->player1);
            vector_pop_back(&queue);
            vector_back(&queue, &s->player2);
            vector_pop_back(&queue);
            s->turn = 0;
            generate_field(s->field1);
            generate_field(s->field2);

            msg_t msg;
            init_data_msg(&msg, s->field1, 100, MT_FIELD, i);
            send_msg(s->player1.sock, &msg);
            close_msg(&msg);
            init_data_msg(&msg, s->field2, 100, MT_FIELD, i);
            send_msg(s->player2.sock, &msg);
            close_msg(&msg);

            int turn = 1;
            init_data_msg(&msg, &turn, sizeof(int), MT_TURN, 0);
            send_msg(s->player1.sock, &msg);
            close_msg(&msg);
            turn = 0;
            init_data_msg(&msg, &turn, sizeof(int), MT_TURN, 0);
            send_msg(s->player2.sock, &msg);
            close_msg(&msg);
        }
    }
    break;
case MT_SHOT:{
        printf("shot\n");

        shot *sh = (shot*)req.buffer;
        int id = req.id;
        printf("session id: %d\n", id);

        if(sessions[id]->player1.sock ==
        client_socks[i] && sessions[id]->turn == 0){
            printf("player1 turn\n");
            msg_t msg;
            if(get_cell(sessions[id]->field2, sh->x,
            sh->y) == CL_SHIP){
                printf("hit: %d-%d\n", sh->x, sh->y);
                set_cell(sessions[id]->field2,
                sh->x, sh->y, CL_DESTROYED);
                if(!check_field(sessions[id]->field2)){
                    init_data_msg(&msg, &id, 1, MT_WIN, id);
                    send_msg(sessions[id]->player1.sock, &msg);
                    req.type = MT_LOSE;
                }
                else{
                    init_data_msg(&msg, &id, 1, MT_HIT, id);
                    send_msg(sessions[id]->player1.sock, &msg);
                    req.type = MT_HIT;
                }
            }
            else{
                printf("miss: %d-%d\n", sh->x, sh->y);
                set_cell(sessions[id]->field2,
                sh->x, sh->y, CL_MISS);
                init_data_msg(&msg, &id, 1, MT_MISS, id);
                send_msg(sessions[id]->player1.sock, &msg);
```

```
                        req.type = MT_MISS;
                    }
                    close_msg(&msg);

                    send_msg(sessions[id]->player2.sock, &req);
                }
                else{
                    printf("player2 turn\n");
                    msg_t msg;
                    if(get_cell(sessions[id]->field1, sh->x, sh->y) == CL_SHIP){
                        printf("hit: %d-%d\n", sh->x, sh->y);
                        set_cell(sessions[id]->field1, sh->x, sh->y, CL_DESTROYED);
                        if(!check_field(sessions[id]->field1)){
                            init_data_msg(&msg, &id, 1, MT_WIN, id);
                            send_msg(sessions[id]->player2.sock, &msg);
                            req.type = MT_LOSE;
                        }
                        else{
                            init_data_msg(&msg, &id, 1, MT_HIT, id);
                            send_msg(sessions[id]->player2.sock, &msg);
                            req.type = MT_HIT;
                        }
                    }
                    else{
                        printf("miss: %d-%d\n", sh->x, sh->y);
                        set_cell(sessions[id]->field1, sh->x, sh->y, CL_MISS);

                        init_data_msg(&msg, &id, 1, MT_MISS, id);
                        send_msg(sessions[id]->player2.sock, &msg);
                        req.type = MT_MISS;
                    }
                    close_msg(&msg);

                    send_msg(sessions[id]->player1.sock, &req);
                }
                int ended = 0;
                if(!check_field(sessions[id]->field2)){
                    printf("player1 won\n");
                    end_session(sessions[id], 0);
                    ended = 1;

                }
                if(!check_field(sessions[id]->field1)){
                    printf("player2 won\n");
                    end_session(sessions[id], 1);
                    ended = 1;
                }

                if(ended){
                    free(sessions[id]);
                    sessions[id] = NULL;
                }
            }
            break;
        case MT_RATING:{
                printf("requested rating\n");
                msg_t msg;
                init_data_msg(&msg, base.data, base.count *
                sizeof(record), MT_RATING, 0);
                send_msg(client_socks[i], &msg);
                close_msg(&msg);
            }
        default:
            break;
    }
```

```
                    }
                }
            }
        }
    }

    printf("releasing resources\n");
    vector_destroy(&queue);
    for(int i = 0; i < MAX_SESSIONS_C; i++)
        if(sessions[i] != NULL)
            free(sessions[i]);

    for(int i = 0; i < MAX_CLIENT_C; i++){
        if(client_socks[i] != 0){
            close(client_socks[i]);
            client_socks[i] = 0;
        }
    }
    close(master_sock);

    printf("saving data\n");
    save(argv[2], &base);
}
```

client.c

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

#include "database.h"
#include "interact.h"
#include "field.h"
#include "render.h"

#define READ_BUFF_L 256

int menu(){
    printf("1: play\n");
    printf("2: rating\n");
    printf("3: exit\n");
    int mode;
    scanf("%d", &mode);
    return mode;
}

void login(int socket, char *nickname){
    msg_t msg;
    init_data_msg(&msg, nickname, strlen(nickname) + 1, MT_LOGIN, 0);
    send_msg(socket, &msg);
    close_msg(&msg);
}

record* get_rating(int socket, int *count){
    msg_t msg;
    init_data_msg(&msg, &msg, 1, MT_RATING, 0);
    send_msg(socket, &msg);
    close_msg(&msg);
    recieve_msg(socket, &msg);
    record *rec = (record*)malloc(msg.size);
```

```
    memcpy(rec, msg.buffer, msg.size);
    *count = msg.size / sizeof(record);
    close_msg(&msg);
    return rec;
}

void print_field(char *field){
    for (int i = 0; i < 10; i++){
        for (int j = 0; j < 10; j++){
            printf("%c ", field[i * 10 + j]);
        }
        printf("\n");
    }
}

void render_ini(buffer *screen){
    int status = init_buffer(screen, 40, 15);
    assert(status > 0);

    char coord[10];
    for (int i = 0; i < 10; i++)
        coord[i] = '0' + i;

    fill_buffer(screen, ' ');

    draw_to_buffer(coord, 10, 1, screen, 2, 1);
    draw_to_buffer(coord, 1, 10, screen, 1, 2);

    draw_to_buffer(coord, 10, 1, screen, 15, 1);
    draw_to_buffer(coord, 1, 10, screen, 14, 2);

    set_background_color(BG_WHITE);
    set_text_color(CH_BLACK);
}

void restore_console(){
    set_background_color(BG_BLACK);
    set_text_color(CH_WHITE);
    printf("\033[2J");
    printf("\033[0;0f");
}

void set_console(){
    set_background_color(BG_WHITE);
    set_text_color(CH_BLACK);
    printf("\033[2J");
    printf("\033[0;0f");
}

int main(int argc, char **argv){
    set_console();
    if(argc < 3){
        printf("usage: client <port>\n");
        exit(EXIT_SUCCESS);
    }

    int port = atoi(argv[1]);

    int sock;
    if((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0){
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    struct sockaddr_in addr;
```

```
addr.sin_family = AF_INET;
if(inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr) < 0){
    perror("address convertion failed");
    exit(EXIT_FAILURE);
}
addr.sin_port = htons(port);

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

char *nickname = argv[2];

database base;

int should_close = 0;
int playing = 0;

buffer screen;

int status = init_buffer(&screen, 40, 15);
assert(status > 0);

render_ini(&screen);

while (!should_close){

    int mode = 0;

    while(mode == 0){
        do{
            mode = menu();
        } while(mode < 1 && mode > 3);

        if(mode == 1){
            login(sock, nickname);
            playing = 1;
            printf("waiting for opponent...\n");
        }
        else if(mode == 2){
            int count = 0;
            record *recs = get_rating(sock, &count);
            printf("nickname:\twinrate:\twin:\tlose:\tleave:\n");
            for(int i = 0; i < count; i++){
                int denum = recs[i].wins + recs[i].losses;
                float winrate = denum == 0 ? 0 : (float)recs[i].wins / denum;
                printf("%s\t\t%2.2f\t\t%d\t%d\t%d\n", recs[i].uname, winrate,
                    recs[i].wins, recs[i].losses, recs[i].lefts);
            }

            mode = 0;
        }
        else if(mode == 3){
            restore_console();
            exit(EXIT_SUCCESS);
        }
    }

    msg_t rep;
    init_msg(&rep);

    do{
        close_msg(&rep);
        recieve_msg(sock, &rep);
```

```
} while (rep.type != MT_FIELD);

int sess_id = rep.id;

printf("session id: %d\n", sess_id);

char field[100];
memcpy(field, rep.buffer, 100);
char op_field[100];
fill_field(op_field, CL_WATER);

do{
    close_msg(&rep);
    recieve_msg(sock, &rep);
} while (rep.type != MT_TURN);

int turn = *(int *)rep.buffer;

while(playing){
    draw_to_buffer(field, 10, 10, &screen, 2, 2);
    draw_to_buffer(op_field, 10, 10, &screen, 15, 2);
    render_r(&screen);

    if(turn){
        while(1){
            shot sh;
            do{
                printf("enter shot coords\n");
                scanf("%d", &sh.x);
                scanf("%d", &sh.y);
            }while(get_cell(op_field, sh.x, sh.y) != CL_WATER);

            msg_t msg;
            init_data_msg(&msg, &sh, sizeof(shot), MT_SHOT, sess_id);
            send_msg(sock, &msg);
            close_msg(&msg);

            close_msg(&rep);
            recieve_msg(sock, &rep);

            if (rep.type == MT_WIN){
                set_cell(op_field, sh.x, sh.y, CL_DESTROYED);
                printf("Congratulation! You won!\n");
                mode = 0;
                playing = 0;
                break;
            }
            if (rep.type == MT_MISS){
                printf("miss\n");
                set_cell(op_field, sh.x, sh.y, CL_MISS);
                turn = 0;
                break;
            }
            printf("hit\n");
            set_cell(op_field, sh.x, sh.y, CL_DESTROYED);
        }
    }
    else{
        while(1){
            msg_t msg;

            recieve_msg(sock, &rep);
            shot *sh = (shot*)rep.buffer;
            if (rep.type == MT_WIN){
                printf("Congratulation! You won!\n");
```

```
                        mode = 0;
                        playing = 0;
                        break;
                    }
                    if (rep.type == MT_LOSE){
                        set_cell(field, sh->x, sh->y, CL_DESTROYED);
                        printf("You lose! Better luck next time!\n");
                        mode = 0;
                        playing = 0;
                        break;
                    }
                    if (rep.type == MT_MISS){
                        printf("miss\n");

                        set_cell(field, sh->x, sh->y, CL_MISS);
                        turn = 1;
                        break;
                    }
                    printf("hit\n");

                    set_cell(field, sh->x, sh->y, CL_DESTROYED);
                }
            }
        }
    }

    restore_console();
}
```

field.h

```
#ifndef FIELD_H
#define FIELD_H

#include <stdlib.h>

#define CL_WATER '~'
#define CL_SHIP '#'
#define CL_DESTROYED 'X'
#define CL_MISS 'o'

#define OR_VERTICAL 0
#define OR_HORIZONTAL 1

char get_cell(char *b, int x, int y){
    return b[y * 10 + x];
}

void set_cell(char *b, int x, int y, char c){
    b[y * 10 + x] = c;
}

void fill_field(char *b, char c){
    for(int i = 0; i < 100; i++)
        b[i] = c;
}

int can_be_ship(char *field, int x, int y){
    for(int i = -1; i <= 1; i++){
        for(int j = -1; j <= 1; j++){
            int cur_x = x + i;
            int cur_y = y + j;

            if(cur_x >= 0 && cur_x < 10 && cur_y >= 0 && cur_y < 10
                && get_cell(field, x + i, y + j) == CL_SHIP){
```

```
                    return 0;
                }
            }
        }
        return 1;
    }

    int can_place(char *field, int x, int y, int dir, int size){
        if(dir == OR_HORIZONTAL){
            for(int i = x; i < x + size; i++)
                if(i >= 10 || !can_be_ship(field, i, y))
                    return 0;
        }
        else{
            for (int i = y; i < y + size; i++)
                if (i >= 10 || !can_be_ship(field, x, i)){
                    return 0;
                }
        }
        return 1;
    }

    void place_ship(char *field, int size){
        int x = rand() % 10;
        int y = rand() % 10;
        int dir = rand() % 2;
        while(!can_place(field, x, y, dir, size)){
            x = rand() % 10;
            y = rand() % 10;
            dir = rand() % 2;
        }
        if(dir == OR_HORIZONTAL){
            for(int i = x; i < x + size; i++)
                set_cell(field, i, y, CL_SHIP);
        }
        else{
            for (int i = y; i < y + size; i++)
                set_cell(field, x, i, CL_SHIP);
        }
    }

    void generate_field(char *field){
        fill_field(field, CL_WATER);
        for(int i = 4; i > 0; i--){
            for(int j = 0; j < 5 - i; j++){
                place_ship(field, i);
            }
        }
    }

    int check_field(char *field){
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(get_cell(field, j, i) == CL_SHIP)
                    return 1;
            }
        }
        return 0;
    }
    #endif
```

database.h

```
#ifndef DATABASE_H
#define DATABASE_H
```

```c
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>
#include <assert.h>
#include <string.h>

#define UNAME_LEN_LIM 20

typedef struct{
    char uname[UNAME_LEN_LIM];
    int wins;
    int losses;
    int lefts;
} record;

typedef struct{
    record *data;
    int count;
}database;

database init(){
    database res = {NULL, 0};
    return res;
}

database load(char *filename){
    struct stat file_stat;

    int fd = open(filename, O_RDONLY);
    assert(fd > 0);

    int status = fstat(fd, &file_stat);
    assert(status == 0);

    database res;
    res.data = (record*)malloc(file_stat.st_size);
    read(fd, res.data, file_stat.st_size);

    res.count = file_stat.st_size / sizeof(record);

    close(fd);
    return res;
}

void save(char *filename, database *base){
    int fd = open(filename, O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    assert(fd > 0);

    write(fd, base->data, base->count * sizeof(record));

    close(fd);
}

void add(database *base, record rec){
    if(base->data != NULL)
        base->data = (record*)realloc(base->data, (base->count + 1) * sizeof(record));
    else
        base->data = (record*)malloc(sizeof(record));
    base->data[base->count] = rec;
    base->count++;
}
```

```
record* get_rec(database *base, int index){
    return &base->data[index];
}

record* find_rec(database *base, char *name){
    for(int i = 0; i < base->count; i++)
        if(!strcmp(name, base->data[i].uname))
            return &base->data[i];
    return NULL;
}

#endif
```

render.h

```
#ifndef RENDER_H
#define RENDER_H

#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <ctype.h>

#include "field.h"

typedef struct{
    int width;
    int height;
    char **data;
} buffer;

#define CH_BLACK "\033[30m"
#define CH_RED "\033[31m"
#define CH_GREEN "\033[32m"
#define CH_BROWN "\033[33m"
#define CH_BLUE "\033[34m"
#define CH_MAGENTA "\033[35m"
#define CH_CYAN "\033[36m"
#define CH_WHITE "\033[37m"

#define BG_BLACK "\033[40m"
#define BG_RED "\033[41m"
#define BG_GREEN "\033[42m"
#define BG_BROWN "\033[43m"
#define BG_BLUE "\033[44m"
#define BG_MAGENTA "\033[45m"
#define BG_CYAN "\033[46m"
#define BG_WHITE "\033[47m"

void set_background_color(const char *c){
    printf("%s", c);
}

void set_text_color(const char *c){
    printf("%s", c);
}

int init_buffer(buffer *b, int w, int h)
{
    b->width = w;
    b->height = h;
    if((b->data = (char**)malloc(h * sizeof(char*))) == NULL){
        return -1;
    }
```

```
    for(int i = 0; i < h; i++){
        if((b->data[i] = (char*)malloc((w + 1) * sizeof(char)))== NULL){
            return -1;
        }
        b->data[i][w] = '\0';
    }
    return 1;
}

void draw_to_buffer(char *f, int w, int h, buffer *b, int x, int y){
    int cur_x, cur_y;
    for(int i = 0; i < h; i++){
        for(int j = 0; j < w; j++){
            cur_x = j + x;
            cur_y = i + y;
            if(cur_y >= 0 && cur_y < b->height && cur_x >= 0 && cur_x < b->width){
                b->data[cur_y][cur_x] = *(f + i * w + j);
            }
        }
    }
}

void fill_buffer(buffer *b, char c){
    for(int i = 0; i < b->height; i++)
        for(int j = 0; j < b->width; j++)
            b->data[i][j] = c;
}

void render_r(buffer *b){
    printf("\e[1;1H\e[2J");
    printf("\033[0;0f");
    for(int i = 0; i < b->height; i++){
        for(int j = 0; j < b->width; j++){
            char c = b->data[i][j];
            if(c == ' ' || (isalpha(c) && c != 'X' && c != 'o') || isdigit(c)){
                set_text_color(CH_BLACK);
                printf("%c ", c);
            }
            else{
                switch (c)
                {
                    case CL_SHIP:
                        set_text_color(CH_BROWN);
                        break;
                    case CL_WATER:
                        set_text_color(CH_BLUE);
                        break;
                    case CL_DESTROYED:
                        set_text_color(CH_RED);
                        break;
                    case CL_MISS:
                        set_text_color(CH_CYAN);
                        break;
                    default:
                        break;
                }
                printf("%c ", c);
            }
        }
        printf("\n");
    }
}

#endif
```

Демонстрация работы

Client1:

```
$ ./client 8888 boar
1: play
2: rating
3: exit
1
waiting for opponent...
session id: 0
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ # # # ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 ~ ~ ~ ~ ~ ~ ~ ~ ~ #      1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ # ~ # ~ ~ ~ ~ #      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ # ~ ~ ~ ~ ~ ~ #      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ ~ ~ ~ ~ ~ ~ ~ #      4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ ~ ~ ~ ~ # ~ ~ ~      5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ # # # ~ ~ ~ ~ ~      6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 # ~ # ~ ~ ~ ~ ~ # ~      8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # # ~ ~ # ~      9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
enter shot coords
0 1
hit
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ # # # ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 ~ ~ ~ ~ ~ ~ ~ ~ ~ #      1 X ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ # ~ # ~ ~ ~ ~ #      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ # ~ ~ ~ ~ ~ ~ #      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ ~ ~ ~ ~ ~ ~ ~ #      4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ ~ ~ ~ ~ # ~ ~ ~      5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ # # # ~ ~ ~ ~ ~      6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 # ~ # ~ ~ ~ ~ ~ # ~      8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # # ~ ~ # ~      9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
enter shot coords
0 3
miss
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ # # # ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 ~ ~ ~ ~ ~ ~ ~ ~ ~ #      1 X ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ # ~ # ~ ~ ~ ~ #      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ # ~ ~ ~ ~ ~ ~ #      3 o ~ ~ ~ ~ ~ ~ ~ ~ ~
```

```
  4 ~ ~ ~ ~ ~ ~ ~ ~ #      4 ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ ~ ~ ~ ~ # ~ ~ ~    5 ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ # # # ~ ~ ~ ~ ~    6 ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~    7 ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 # ~ # ~ ~ ~ ~ ~ # ~    8 ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # # ~ ~ # ~    9 ~ ~ ~ ~ ~ ~ ~ ~ ~
hit
    0 1 2 3 4 5 6 7 8 9      0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ X # # ~ ~ ~    0 ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 ~ ~ ~ ~ ~ ~ ~ ~ ~ #    1 X ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ # ~ # ~ ~ ~ ~ #    2 ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ # ~ ~ ~ ~ ~ ~ #    3 o ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ ~ ~ ~ ~ ~ ~ ~ #    4 ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ ~ ~ ~ ~ # ~ ~ ~    5 ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ # # # ~ ~ ~ ~ ~    6 ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~    7 ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 # ~ # ~ ~ ~ ~ ~ # ~    8 ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # # ~ ~ # ~    9 ~ ~ ~ ~ ~ ~ ~ ~ ~
miss
    0 1 2 3 4 5 6 7 8 9      0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ X # # ~ ~ ~    0 ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 ~ ~ ~ ~ ~ ~ ~ ~ ~ #    1 X ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ # ~ # ~ ~ ~ ~ #    2 ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ # ~ ~ ~ ~ ~ ~ #    3 o ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ ~ ~ o ~ ~ ~ ~ #    4 ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ ~ ~ ~ ~ # ~ ~ ~    5 ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ # # # ~ ~ ~ ~ ~    6 ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~    7 ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 # ~ # ~ ~ ~ ~ ~ # ~    8 ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # # ~ ~ # ~    9 ~ ~ ~ ~ ~ ~ ~ ~ ~
enter shot coords
^C
$
```

Client2:

```
$ ./client 8888 wolf
1: play
2: rating
3: exit
1
waiting for opponent...
session id: 0
    0 1 2 3 4 5 6 7 8 9      0 1 2 3 4 5 6 7 8 9
```

```
  0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 # ~ # # # ~ # # ~ #      1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ ~ ~ ~ # # # # ~      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ # ~ ~ ~ ~ ~ ~ ~      4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ # ~ ~ ~ ~ ~ ~ ~      5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 # ~ ~ ~ ~ ~ # ~ # ~      7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 ~ ~ ~ ~ # ~ ~ ~ # ~      8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # ~ ~ ~ # ~      9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
hit
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 X ~ # # # ~ # # ~ #      1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 ~ ~ ~ ~ ~ # # # # ~      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ # ~ ~ ~ ~ ~ ~ ~      4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ # ~ ~ ~ ~ ~ ~ ~      5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 # ~ ~ ~ ~ ~ # ~ # ~      7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 ~ ~ ~ ~ # ~ ~ ~ # ~      8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # ~ ~ ~ # ~      9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
miss
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  1 X ~ # # # ~ # # ~ #      1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 o ~ ~ ~ ~ # # # # ~      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ # ~ ~ ~ ~ ~ ~ ~      4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ # ~ ~ ~ ~ ~ ~ ~      5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 # ~ ~ ~ ~ ~ # ~ # ~      7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 ~ ~ ~ ~ # ~ ~ ~ # ~      8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # ~ ~ ~ # ~      9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
enter shot coords
4 0
hit
    0 1 2 3 4 5 6 7 8 9        0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      0 ~ ~ ~ ~ X ~ ~ ~ ~ ~
  1 X ~ # # # ~ # # ~ #      1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~      2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 o ~ ~ ~ ~ # # # # ~      3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
```

```
  4 ~ ~ # ~ ~ ~ ~ ~ ~ ~     4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  5 ~ ~ # ~ ~ ~ ~ ~ ~ ~     5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~     6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 # ~ ~ ~ ~ ~ # ~ # ~     7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 ~ ~ ~ ~ # ~ ~ ~ # ~     8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # ~ ~ ~ # ~     9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
enter shot coords
4 4
miss
    0 1 2 3 4 5 6 7 8 9       0 1 2 3 4 5 6 7 8 9
  0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~     0 ~ ~ ~ ~ X ~ ~ ~ ~ ~
  1 X ~ # # # ~ # # ~ #     1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~     2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  3 o ~ ~ ~ ~ # # # # ~     3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  4 ~ ~ # ~ ~ ~ ~ ~ ~ ~     4 ~ ~ ~ ~ o ~ ~ ~ ~ ~
  5 ~ ~ # ~ ~ ~ ~ ~ ~ ~     5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~     6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  7 # ~ ~ ~ ~ ~ # ~ # ~     7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  8 ~ ~ ~ ~ # ~ ~ ~ # ~     8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
  9 ~ ~ ~ ~ # ~ ~ ~ # ~     9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~
Congratulation! You won!
1: play
2: rating
3: exit
3
$
```

```
Server:
$./server 8888 data.dat
port: 8888
new connection
connected: 4
new connection
connected: 5
processing client: 5
login boar
player boar not found
added to database
statistics:
        wins: 0
        losses: 0
        lefts: 0
queued
```

```
processing client: 4
login wolf
player wolf not found
added to database
statistics:
        wins: 0
        losses: 0
        lefts: 0
queued
start new session
processing client: 5
shot
session id: 0
player1 turn
hit: 0-1
processing client: 5
shot
session id: 0
player1 turn
miss: 0-3
processing client: 4
shot
session id: 0
player2 turn
hit: 4-0
processing client: 4
shot
session id: 0
player2 turn
miss: 4-4
processing client: 5
disconnected
session ended because of disconnected player
before closing
processing client: 4
disconnected
before closing
```

Strace

```
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
setsockopt(3, SOL_SOCKET, SO_REUSEPORT, [1], 4) = 0
```

```
bind(3, {sa_family=AF_INET, sin_port=htons(8888), sin_addr=inet_addr("0.0.0.0")}, 16) =
listen(3, 3)                            = 0
select(4, [3], NULL, NULL, NULL)        = 1 (in [3])
write(1, "new connection\n", 15)        = 15
accept(3, NULL, NULL)                   = 4
write(1, "connected: 4\n", 13)          = 13
select(5, [3 4], NULL, NULL, NULL)      = 1 (in [3])
write(1, "new connection\n", 15)        = 15
accept(3, NULL, NULL)                   = 5
write(1, "connected: 5\n", 13)          = 13
select(6, [3 4 5], NULL, NULL, NULL)    = 1 (in [5])
write(1, "processing client: 5\n", 21)  = 21
recvfrom(5, "\5\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(5, "\1\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(5, "\0\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(5, "boar\0", 5, 0, NULL, NULL) = 5
write(1, "login boar\n", 11)            = 11
...
select(6, [3 4 5], NULL, NULL, NULL)    = 1 (in [4])
write(1, "processing client: 4\n", 21)  = 21
recvfrom(4, "\5\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(4, "\1\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(4, "\0\0\0\0", 4, 0, NULL, NULL) = 4
recvfrom(4, "wolf\0", 5, 0, NULL, NULL) = 5
write(1, "login wolf\n", 11)            = 11
...
sendto(5, "d\0\0\0", 4, 0, NULL, 0)     = 4
sendto(5, "\3\0\0\0", 4, 0, NULL, 0)    = 4
sendto(5, "\0\0\0\0", 4, 0, NULL, 0)    = 4
sendto(5, "~~~~###~~~~~~~~~~~~#~~#~#~~~~#~~"..., 100, 0, NULL, 0) = 100
sendto(4, "d\0\0\0", 4, 0, NULL, 0)     = 4
sendto(4, "\3\0\0\0", 4, 0, NULL, 0)    = 4
sendto(4, "\0\0\0\0", 4, 0, NULL, 0)    = 4
sendto(4, "~~~~~~~~~~#~###~##~#~~~~~~~~~~~"..., 100, 0, NULL, 0) = 100
sendto(5, "\4\0\0\0", 4, 0, NULL, 0)    = 4
sendto(5, "\4\0\0\0", 4, 0, NULL, 0)    = 4
sendto(5, "\0\0\0\0", 4, 0, NULL, 0)    = 4
sendto(5, "\1\0\0\0", 4, 0, NULL, 0)    = 4
...
select(6, [3 4 5], NULL, NULL, NULL)    = 1 (in [5])
write(1, "processing client: 5\n", 21)  = 21
recvfrom(5, "", 4, 0, NULL, NULL)       = 0
```

```
write(1, "disconnected\n", 13)          = 13
sendto(4, "\1\0\0\0", 4, 0, NULL, 0)    = 4
sendto(4, "\7\0\0\0", 4, 0, NULL, 0)    = 4
sendto(4, "\0\0\0\0", 4, 0, NULL, 0)    = 4
sendto(4, "4", 1, 0, NULL, 0)           = 1
write(1, "session ended because of disconn"..., 45) = 45
write(1, "before closing\n", 15)        = 15
close(5)                                = 0
select(5, [3 4], NULL, NULL, NULL)      = 1 (in [4])
write(1, "processing client: 4\n", 21)  = 21
recvfrom(4, "", 4, 0, NULL, NULL)       = 0
write(1, "disconnected\n", 13)          = 13
write(1, "before closing\n", 15)        = 15
close(4)                                = 0
select(4, [3], NULL, NULL, NULL)        = ?
ERESTARTNOHAND (To be restarted if no handler)
select(4, [3], NULL, NULL, NULL)        = ?
ERESTARTNOHAND (To be restarted if no handler)
--- SIGINT {si_signo=SIGINT, si_code=SI_KERNEL} ---
rt_sigreturn({mask=[]})                 = -1
EINTR (Interrupted system call)
write(1, "releasing resources\n", 20)   = 20
close(3)                                = 0
write(1, "saving data\n", 12)           = 12
openat(AT_FDCWD, "data.dat", O_WRONLY|O_CREAT, 0600) = 3
write(3, "boar\0\0\0\0\334Y\0\0\0\0\0\0\4\0\0\0\0\0\0\0\1\0\0\0\1\0\0\0"..., 64)
= 64
close(3)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

Выводы

В результате был разработан вполне работоспособный прототип клиент-серверной игры «Морской бой», для разработки которого было применено многое из того, что было получено в ходе изучения курса. В частности организация клиент-серверного взаимодействия с помощью «голых» сокетов, попытка организации логики импровизированной базы данных с помщью `mmap`'а (оказалось излишним, но факт использования был).

Также в процессе разаработки была попытка организовать программу как можно более удобным для процесса разработки и внесения правок образом. Ее можно, по моему мнению, считать достаточно удачной. Ранее при создании чего-то мало-мальски крупного к концу разработки становилось мучительно больно добавлять что-то новое или вносить правки из-за ошибки в проектировании. Хотя код получился довольно объ-

емным, чем мог бы быть, но в целом оказался более удачно организованным и довольно лаконичным.

Приобретенные в процессе прохождения курса «Операционных систем» знания окажутся весьма полезными в дальнейшем и точно не останутся лежать на полке без дела.