Московский авиационный институт (национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Отчет по курсовому проекту по курсу «Операционные системы»

Студент: Шевчук П.В.
Преподаватель: Соколов А.А.
Группа: М8О-206Б
Дата:
Подпись:

Москва, 2017

Задание

Создать собственную игру более, чем для одного пользователя. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.
Игру реализовать на основе любой из выбранных технологий:

- Pipes
- Sockets
- Сервера очередей
- И другие

Описание

Для реализации связи клиент-сервер, как и в 6-8 лабораторных работах, был выбран паттерн RequestResponse и клиент сообщений ZeroMQ. Клиент отправляет запрос на сервер и ждет ответа. После того, как ответ пришел, клиент может продолжать работу. Клиент исполняет роль игрока, а сервер - игровой комнаты.

Количество игроков для конкретного сеанса задается ключом при запуске сервера. Сервер устанавливается по заданному адресу и выводит таблицу игроков, где указаны имена игроков и их статистика игр. Далее задается размер костей и их количество и сервер ждет подключения всех игроков. При запуске клиента игрок задает ключом свой ник и затем вводит номер комнаты. Когда все игроки подключились начинается игра. Каждому игроку выводятся правила текущей игры - условия победы, количество костей и их размер. Чтобы кинуть кость нужно нажать любую клавишу. Когда все игроки совершили необходимое количество бросков на сервере выводится обновленная таблица. Теперь каждый игрок может посмотреть результаты текущей игры, статистику всех игр или топ-3 игроков.

Системные вызовы:

void* zmq_ctx_new(); - создает новый контекст

void exit(int status); – функция выходит из процесса с заданным статусом.

void *zmq socket(void *context, int type); – создает сокет типа type из контекста context. int zmq msg

send(zmq msg t *msg, void *socket, int flags); – отправляет сообщение msg в socket с параметрами flags, возвращает количество отправленных байт, в случае ошибки возвращает -1.

int zmq bind(void *socket, const char *endpoint); – присоединяет socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.

int zmq connect(void *socket, const char *endpoint); – подключает socket к пути endpoint, 0 в случае успеха, -1 в случае ошибки.

int zmq_recv (void * socket , void * buf , size_t len , int flags) - Функция zmq_recv () должна получать сообщение из сокета, на которое ссылается аргумент сокета, и хранить его в буфере, на который ссылается аргумент buf . Любые байты, превышающие длину, заданные аргументом len, должны быть усечены. в случае ошибки возвращает -1.

int zmq_send (void * socket , zmq_msg_t * msg , int flags); - должна поставить в очередь сообщение, на которое ссылается аргумент msg, который должен быть отправлен в сокет, на который ссылается аргумент socket. в случае ошибки возвращает -1.

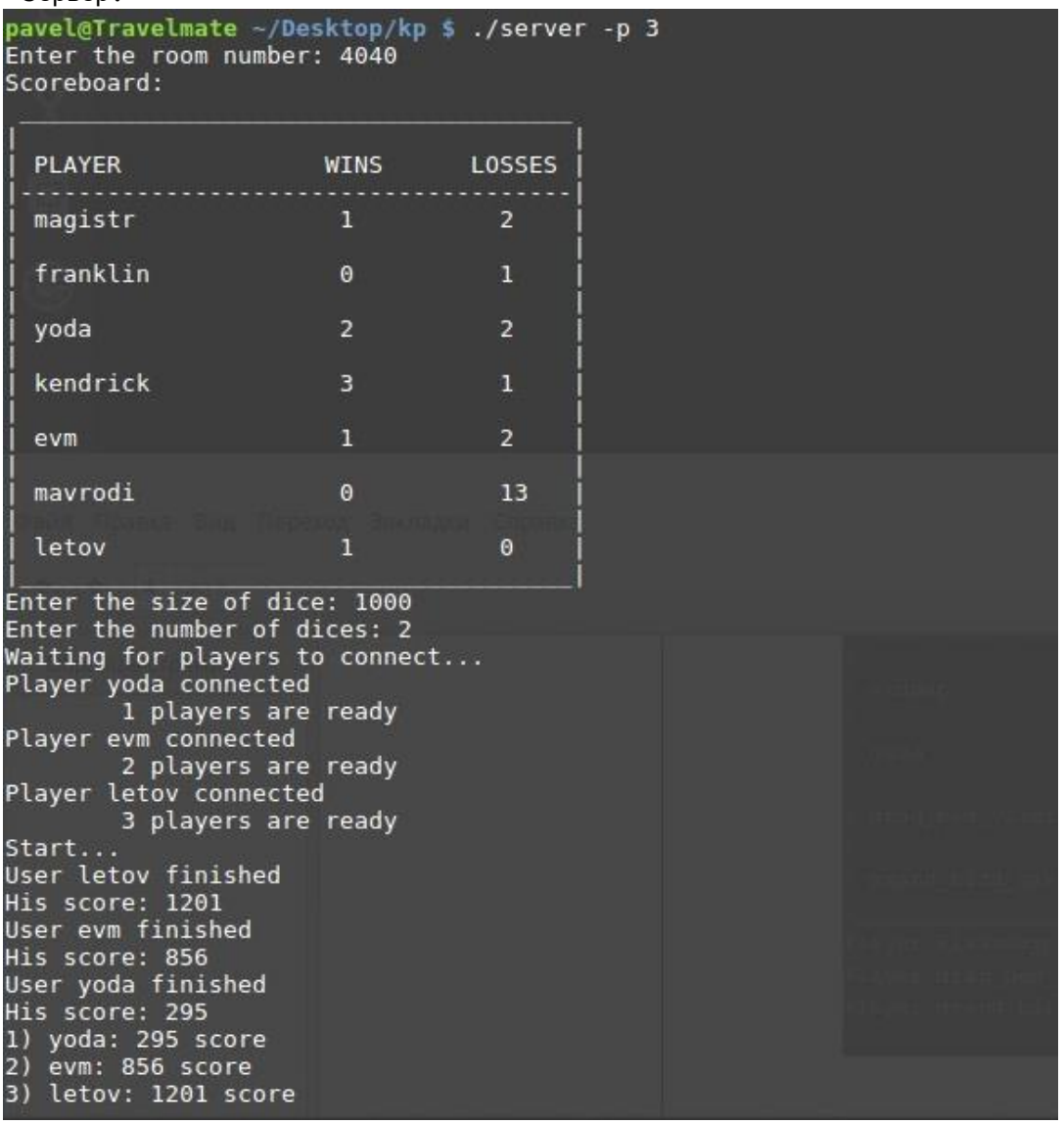
int zmq close(void *socket); – закрывает сокет, возвращает 0 в случае успеха и -1 в случае неудачи.

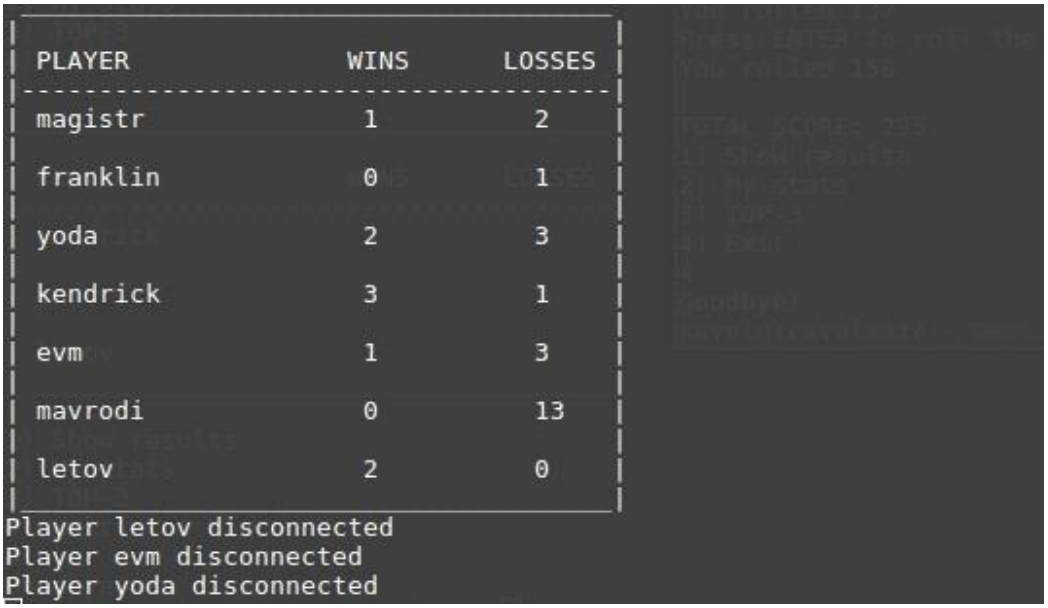
int zmq_ctx_term (void *context); - должна уничтожить контекст ZMQ, возвращает 0 в случае успеха и -1 случае неудачи.

int zmq ctx destroy(void *context); – разрушает контекст context, блокирует доступ всем операциям кроме zmq close, все сообщения в сокетах либо физически отправлены, либо "висят".

Тесты

Сервер:





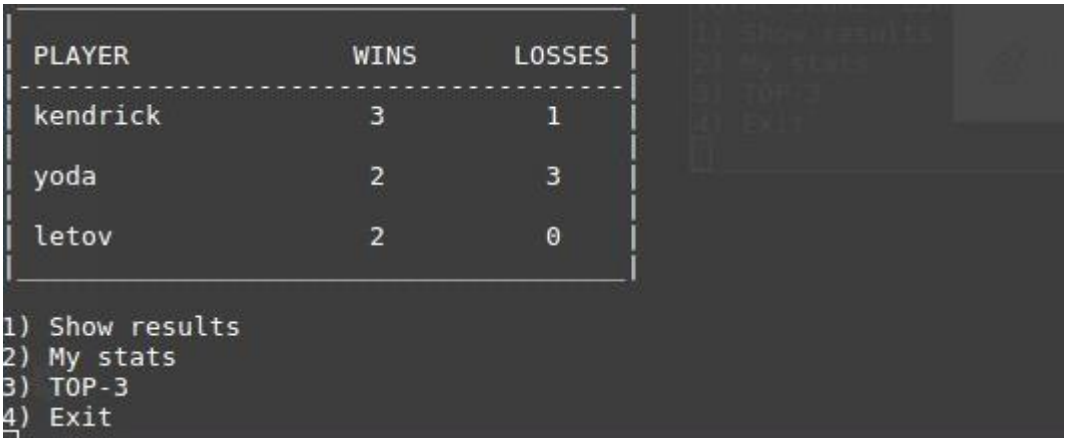
PLAYER	WINS	LOSSES
magistr	1	2
franklin	0	1
yoda	2	3
kendrick	3	1
evm	1	3
mavrodi	0	13
letov	2	0

```
Player letov disconnected
Player evm disconnected
Player yoda disconnected
```

Клиент 1:

```
pavel@Travelmate ~/Desktop/kp $ ./client -l letov
Please enter the room number: 4040
Connect to tcp://localhost:4040
 ================================================
                    RULES:

     THERE ARE 2 DICES OF SIZE 1000 TO BE ROLLED

   THE ONE TO SCORE THE MOST POINTS IS THE WINNER

                  GOOD LUCK!!!
================================================

Press ENTER to roll the dice
You rolled 780
Press ENTER to roll the dice
You rolled 421

TOTAL SCORE: 1201
1) Show results
2) My stats
3) TOP-3
4) Exit
1
1) yoda: 295 score
2) evm: 856 score
3) letov: 1201 score

1) Show results
2) My stats
3) TOP-3
4) Exit
2
Username: letov
Win: 2
Lose: 0

1) Show results
2) My stats
3) TOP-3
4) Exit
3
```

PLAYER	WINS	LOSSES



Клиент 2:

```
pavel@Travelmate ~/Desktop/kp $ ./client -l yoda
Please enter the room number: 4040
Connect to tcp://localhost:4040
 ================================================
                    RULES:

     THERE ARE 2 DICES OF SIZE 1000 TO BE ROLLED

   THE ONE TO SCORE THE MOST POINTS IS THE WINNER

                  GOOD LUCK!!!
================================================

Press ENTER to roll the dice
You rolled 137
Press ENTER to roll the dice
You rolled 158

TOTAL SCORE: 295
1) Show results
2) My stats
3) TOP-3
4) Exit
4
Goodbye!
pavel@Travelmate ~/Desktop/kp $
```

Клиент 3:

```
pavel@Travelmate ~/Desktop/kp $ ./client -l evm
Please enter the room number: 4040
Connect to tcp://localhost:4040
 ================================================
                    RULES:

     THERE ARE 2 DICES OF SIZE 1000 TO BE ROLLED

   THE ONE TO SCORE THE MOST POINTS IS THE WINNER

                  GOOD LUCK!!!
================================================

Press ENTER to roll the dice
You rolled 165
Press ENTER to roll the dice
You rolled 691

TOTAL SCORE: 856
1) Show results
2) My stats
3) TOP-3
4) Exit
4
Goodbye!
pavel@Travelmate ~/Desktop/kp $
```

Код

SERVER.C

```
#include "argument.h"
#include "stats.h"


void Sorting(Result* sb, int n);
void PrintResults(Result* tmp, int n);
void GetTop(Scoreboard* db, char *ans);

int main(int argc, char **argv)
{
    int players_cnt;
    int id[10] = {0};
    int room_number;
    char address[25];
    char answer[2048];
    Result table[10];
    Args tmp;
    Args data[10];

    if (argc == 3 && !strcmp(argv[1],"-p"))
    {
        players_cnt = atoi(argv[2]);
    }
    else
    {
        printf("usage: ./server -p <number of players [2-10]>\n");
        return 0;
    }

    if (players_cnt < 2 || players_cnt > 10)
    {
        printf("usage: ./server -p <number of players [2-10]>\n");
        return 0;
    }
```

```
void* context = zmq_ctx_new();

if (!context)
{
    perror("zmq_ctx_new");
    exit(1);
}

void* responder = zmq_socket(context, ZMQ_REP);

printf("Enter the room number: ");
scanf("%d", &room_number);

sprintf(address, "%s%d", "tcp://*:", room_number);
int bind = zmq_bind(responder, address);

if (bind)
{
    perror("bind");
    exit(1);
}

Scoreboard* db = Create();
FILE *fp = fopen("SB", "r+");

if (Load(&db, fp))
{
    perror("db load");
    exit(1);
}

printf("Scoreboard:\n");
Print(db);

fseek(fp, 0, SEEK_SET);

int dice_number = 0;
int dice_size = 0;
printf("Enter the size of dice: ");
scanf("%d", &dice_size);
if (dice_size < 1)
{
    printf("use  the size of dice > 1\n");
    return 0;
}

printf("Enter the number of dices: ");
scanf("%d", &dice_number);

if (dice_number < 2)
{
    printf("use  the number of dices > 2\n");
    return 0;
}

printf("Waiting for players to connect...\n");

for (int i = 0; i < players_cnt; ++i)
{
    zmq_recv(responder, &tmp, sizeof(Args), 0);

    if (!tmp.players)
    {
        printf("Player %s connected\n", tmp.log);
        tmp.checked = i;
```

```
            table[i].threw = 0;
            tmp.players = players_cnt;
            printf("\t%d players are ready\n", i + 1);
        }
        else
        {
            --i;
        }

        tmp.dice_size = dice_size;
        tmp.dice_number = dice_number;
        zmq_send(responder, &tmp, sizeof(Args), 0);
    }

    int num = players_cnt;

    printf("Start...\n");

    while (num)
    {
        zmq_recv(responder, &tmp, sizeof(Args), 0);

        if (!id[tmp.checked])
        {
            --num;
            tmp.status = 1;
            id[tmp.checked] = 1;
        }

        zmq_send(responder, &tmp, sizeof(Args), 0);
    }

    for (int i = 0; i < players_cnt; ++i)
    {
        id[i] = 0;
    }

    num = players_cnt;

    int data_cnt = 0;

    while (1)
    {
        if (!players_cnt)
        {
            for (int i = 1; i < num; ++i)
            {
                for (int j = i; j > 0; --j)
                {
                    if (data[j - 1].result > data[j].result)
                    {
                        Args tmp = data[j - 1];
                        data[j - 1] = data[j];
                        data[j] = tmp;
                    }
                }
            }

            for (int i = 0; i < num - 1; ++i) // проигравшие
            {
                Update(db, data[i].log, 0);
            }

            Update(db, data[num - 1].log, 1); // выигравший

            Sorting(table, num);
```

```
            PrintResults(table, num);

            table[0].threw = 1;
            players_cnt = 1;

            Print(db);
            Save(db, fp);

            fclose(fp);
        }

        zmq_recv(responder, &tmp, sizeof(Args), 0);

        switch (tmp.status)
        {
            case 0:
            {
                if (!id[tmp.checked])
                {
                    strcpy(data[data_cnt].log, tmp.log);
                    data[data_cnt].result = tmp.result;
                    ++data_cnt;

                    strcpy(table[tmp.checked].name, tmp.log);
                    table[tmp.checked].score = tmp.result;
                    id[tmp.checked] = 1;

                    printf("User %s finished\n", tmp.log);
                    printf("His score: %d\n", tmp.result);
                    zmq_send(responder, &answer, sizeof(answer), 0);

                    --players_cnt;
                }

                break;
            }

            case 1:
            {
                if (players_cnt)
                {
                    zmq_send(responder, &table, sizeof(table), 0);
                }
                else
                {
                    table[0].threw = 1;
                    zmq_send(responder, &table, sizeof(table), 0);
                }

                break;
            }

            case 2:
            {
                Player* print;
                print = Find(db, tmp.log);
                sprintf(answer, "Username: %s\nWin: %d\nLose: %d\n", print-
>log, print->wins, print->losses);
                zmq_send(responder, &answer, sizeof(answer), 0);

                break;
            }

            case 3:
            {
                GetTop(db, answer);
```

```
                zmq_send(responder, &answer, 2 * sizeof(answer), 0);

                break;
            }

            case 4:
            {
                printf("Player %s disconnected\n", tmp.log);
                strcpy(answer, "Goodbye!");
                zmq_send(responder, &answer, sizeof(answer), 0);

                break;
            }

            default:
            {
                tmp.players = 0;
                zmq_send(responder, &tmp, sizeof(Args), 0);

                break;
            }
        }

    }

    return 0;
}

void Sorting(Result *sb, int n)
{
    Result tmp;

    for (int i = 1; i < n; ++i)
    {
        for (int j = i; j > 0; --j)
        {
            if (sb[j - 1].score > sb[j].score)
            {
                tmp = sb[j - 1];
                sb[j - 1] = sb[j];
                sb[j] = tmp;
            }
        }
    }
}

void PrintResults(Result* tmp, int n)
{
    for (int i = 0; i < n; ++i)
    {
        printf("%d) %s: %d score\n", i + 1, tmp[i].name, tmp[i].score);
    }
}

void GetTop(Scoreboard* db, char *ans)
{
    Sort(db);
    strcpy(ans, " _____________________________________\n");
    sprintf(ans, "%s|                                    |\n", ans);
    sprintf(ans, "%s| PLAYER              WINS      LOSSES |\n", ans);
    int size = (db->size > 3) ? 3 : db->size;

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

```
            sprintf(ans, "%s|--------------------------------------|\n",
ans);
        }
        else
        {
            sprintf(ans, "%s|                                      |\n",
ans);
        }

        sprintf(ans, "%s| %-20s %-2d         %-2d   |\n", ans, db-
>players[i].log, db->players[i].wins, db->players[i].losses);
    }

    sprintf(ans, "%s|______________________________________|\n", ans);
}
```

CLIENT.C

```
#include "argument.h"
#include <stdlib.h>

void PrintResults(Result *tmp, int n);

int main(int argc, char **argv)
{
    srand((unsigned int) time(NULL));
    char answer[2048];
    Result table[10];
    int room_number;
    char address[25];
    void* context = zmq_ctx_new();

    if (!context)
    {
        perror("zmq_ctx_new");
        exit(1);
    }

    void* requester = zmq_socket(context, ZMQ_REQ);

    Args my_game;
    my_game.players = 0;
    my_game.status = -1;

    if (argc == 3 && !strcmp(argv[1], "-l"))
    {
        strcpy(my_game.log, argv[2]);
    }
    else
    {
        printf("usage: ./client -l <login>\n");
        return 0;
    }

    printf("Please enter the room number: ");
    scanf("%d", &room_number);
    getchar();

    sprintf(address, "%s%d", "tcp://localhost:", room_number);

    int cn = zmq_connect(requester, address);
    printf("Connect to tcp://localhost:%d\n", room_number);

    if (cn)
    {
        perror("connect");
        exit(1);
```

```
    }

    zmq_send(requester, &my_game, sizeof(Args), 0);
    zmq_recv(requester, &my_game, sizeof(Args), 0);

    if (my_game.players == 0)
    {
        printf("No free places\n");
        return 0;
    }

    while (1)
    {
        zmq_send(requester, &my_game, sizeof(Args), 0);
        zmq_recv(requester, &my_game, sizeof(Args), 0);

        if (my_game.status == 1)
        {
            break;
        }
    }

    printf(" ================================================\n");
    printf("                      RULES:                    \n\n");
    printf("     THERE ARE %d DICES OF SIZE %d TO BE ROLLED\n\n",
my_game.dice_number, my_game.dice_size);
    printf("   THE ONE TO SCORE THE MOST POINTS IS THE WINNER\n\n");
    printf("                   GOOD LUCK!!!\n");
    printf("================================================\n\n");

    my_game.result = 0;
    int cur_roll = 0;

    for (int i = 0; i < my_game.dice_number; ++i)
    {
        printf("Press ENTER to roll the dice ");
        getchar();

        cur_roll = rand() % my_game.dice_size;
        printf("You rolled %d\n", cur_roll);

        my_game.result += cur_roll;
    }

    printf("\nTOTAL SCORE: %d\n", my_game.result);

    my_game.status = 0;
    zmq_send(requester, &my_game, sizeof(Args), 0);
    zmq_recv(requester, &answer, sizeof(answer), 0);

    // printf("%s\n", answer);

    while (1)
    {
        int cmd;
        printf("1) Show results\n");
        printf("2) My stats\n");
        printf("3) TOP-3 \n");
        printf("4) Exit\n");
        scanf("%d", &cmd);

        switch (cmd)
        {
            case 1:
            {
                my_game.status = cmd;
```

```
                zmq_send(requester, &my_game, sizeof(Args), 0);
                zmq_recv(requester, &table, sizeof(Result) * 10, 0);

                if (table[0].threw)
                {
                    PrintResults(table, my_game.players);
                }

                break;
            }

            case 2:
            {
                my_game.status = cmd;
                zmq_send(requester, &my_game, sizeof(Args), 0);
                zmq_recv(requester, &answer, sizeof(answer), 0);
                printf("%s\n", answer);

                break;
            }

            case 3:
            {
                my_game.status = cmd;
                zmq_send(requester, &my_game, sizeof(Args), 0);
                zmq_recv(requester, &answer, sizeof(answer), 0);
                printf("%s\n", answer);

                break;
            }

            case 4:
            {
                my_game.status = cmd;
                zmq_send(requester, &my_game, sizeof(Args), 0);
                zmq_recv(requester, &answer, sizeof(answer), 0);
                printf("%s\n", answer);

                zmq_close(requester);
                zmq_ctx_term(context);
                zmq_ctx_destroy(context);

                return 0;

                break;
            }

            default:
            {
                printf("ERROR: Invalid command\n");

                break;
            }
        }
    }

    return 0;
}

void PrintResults(Result *tmp, int n)
{
    for (int i = 0; i < n; ++i)
    {
        printf("%d) %s: %d score\n", i + 1, tmp[i].name, tmp[i].score);
    }
```

```
    printf("\n");
}
```

STATS.C

```
#include "stats.h"

Scoreboard* Create(void)
{
    Scoreboard* pl = (Scoreboard*) malloc(sizeof(Scoreboard));

    if (!pl)
    {
        fprintf(stderr, "ERROR: no memory\n");
        exit(1);
    }

    pl->players = (Player*) malloc(sizeof(Player) * 10);
    pl->size = 0;
    pl->freespace = 10;

    return pl;
}

int Load(Scoreboard** pl, FILE* file)
{
    Player tmp;

    if (file < 0)
    {
        printf("Cannot open file\n");
        return 1;
    }

    while (fscanf(file, "%s %d %d", tmp.log, &tmp.wins, &tmp.losses) == 3) {
        Add((*pl), tmp.log);
        Fill((*pl), tmp.log, tmp.wins, tmp.losses);
    }
    return 0;
}

void Add(Scoreboard* pl, char *log)
{
    strcpy(pl->players[pl->size].log, log);
    pl->players[pl->size].wins = 0;
    pl->players[pl->size].losses = 0;
    pl->size++;
    pl->freespace--;
}

void Fill(Scoreboard* pl, char *log, int win, int lose)
{
    Player* toFill = Find(pl, log);

    if (toFill)
    {
        toFill->wins = win;
        toFill->losses = lose;
    }
}

Player* Find(Scoreboard* pl, char *log)
{
    if (pl)
    {
        for (int i = 0; i < pl->size; ++i)
        {
```

```
            if (!strcmp(pl->players[i].log, log))
            {
                return &(pl->players[i]);
            }
        }
    }

    return NULL;
}

void Print(Scoreboard* pl)
{
    if (pl)
    {
        printf(" _____\n");
        printf("|                                        |\n");
        printf("| PLAYER              WINS      LOSSES |\n");

        for (int i = 0; i < pl->size; ++i)
        {
            if (i == 0)
            {
                printf("|----------------------------------------|\n");
            }
            else
            {
                printf("|                                        |\n");
            }

            printf("| %-20s %-2d         %-2d   |\n", pl->players[i].log,
pl->players[i].wins, pl->players[i].losses);
        }

        printf("|_____|\n");
    }
}

void Update(Scoreboard* pl, char* log, int res)
{
    Player* player = Find(pl, log);

    if (player == NULL)
    {
        Add(pl, log);
        player = Find(pl, log);
    }

    res == 1 ? player->wins++ : player->losses++;
}



int Save(Scoreboard* pl, FILE* file)
{
    if (file < 0)
    {
        printf("Cannot open file\n");
        return 1;
    }

    for (int i = 0; i < pl->size; ++i)
    {
        fprintf(file, "%s %d %d ", pl->players[i].log, pl->players[i].wins,
pl->players[i].losses);
    }
```

```
    return 0;
}
void Destroy(Scoreboard** pl)
{
    free((*pl)->players);
    (*pl)->players = NULL;

    free(*pl);
    *pl = NULL;
}

void Sort(Scoreboard* pl)
{
    struct _Player tmp;

    if (pl)
    {
        for (int i = 1; i < pl->size; ++i)
        {
            for (int j = i; j > 0; --j)
            {
                if (pl->players[j - 1].wins < pl->players[j].wins)
                {
                    tmp = pl->players[j - 1];
                    pl->players[j - 1] = pl->players[j];
                    pl->players[j] = tmp;
                }
            }
        }
    }
}
```

Выводы

Я углубил и закрепил свои навыки работы с системными вызовами, освоил клиент сообщений ZeroMQ, который оказался очень удобным средством для взаимодействия между процессами. Игра простая, но при желании ее можно усложнить, поскольку для этого требуется лишь изменить передаваемые данные, а механизм передачи и устройство комнат можно оставить как есть.