Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования "Новгородский государственный университет имени Ярослава Мудрого"

Кафедра «Информационных технологий и систем»

Дисциплина «Операционные системы»

Курсовая работа

«Три пальца»

Выполнил студент группы 9091

\_\_\_\_\_\_\_\_\_\_\_\_\_/Лопатин Даниил Михайлович/

Подпись ФИО

Принял преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_/Ананьев Владислав Валерьевич/

Подпись ФИО

Великий Новгород

2021

ОГЛАВЛЕНИЕ

[Разработка проекта 2](#_Toc75237448)

[Введение 2](#_Toc75237449)

[Постановка задачи 2](#_Toc75237450)

[Выбор инструментальных средств 3](#_Toc75237451)

[Выбор модели 3](#_Toc75237452)

[Выбор протокола транспортной OSI-модели 3](#_Toc75237453)

[Результат работы 4](#_Toc75237454)

[Заключение 4](#_Toc75237455)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 4](#_Toc75237456)

[ПРИЛОЖЕНИЕ А 5](#_Toc75237457)

[ЛИСТИНГ SERVER.C 5](#_Toc75237458)

[ЛИСТИНГ CLIENT.C 21](#_Toc75237459)

[ЛИСТИНГ UTILITY.H 28](#_Toc75237460)

[ЛИСТИНГ UTILITY.C 29](#_Toc75237461)

# Разработка проекта

## Введение

Целью данной работы является создания клона игры «Три пальца». Три пальца — это игра для двух участников и ведущего, в которой игроки одновременно выбирают числа пальцев (от 1 до 3 включительно) и эти числа суммируются ведущим в качестве промежуточного количества очков. Если эта сумма четная, раунд выигрывает второй игрок, иначе – первый, о чем сигнализирует ведущий. Выигравший игрок прибавляет промежуточную сумму очков к своим очкам, а проигравший – отнимает. Выигрывает тот игрок, чье количество очков на момент завершения последнего раунда больше.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux.

Основные правила и суть задачи. Количества игроков в данной игре — 2 игрока, причем должно быть 2 пары: игрок против игрока и игрок против компьютера.

В том случае, если запускается режим игрок против игрока, в начале игры игроками задаются количества раундов. Если эти количества не совпадают, один из игроков в случайном порядке получает приоритет на согласие/несогласие с количеством раундов оппонента. При согласии первого игрока игра продолжается с количеством раундов оппонента; иначе – первый игрок задает новое количество раундов, приоритет на согласие/несогласие с новым количеством раундов передается второму игроку, если второй игрок согласен с новым количеством раундов – игра продолжается с этим количеством, в противном случае происходит завершение программы с выводом соответствующей ошибки.

Ход игрока состоит в выборе числа пальцев (от 1 до 3 включительно). За обработку хода каждого игрока отвечает отдельный поток.

Ход ведущего состоит в суммировании промежуточного общего числа пальцев двух игроков в данном раунде, определении на его основе выигравшего и проигравшего раунд игроков, а также определении выигравшего и проигравшего партию игроков в том случае, если данный раунд является последним. За обработку хода ведущего отвечает отдельный поток.

Обязательное условие работы потоков двух игроков (не имеет значения, двух реальных игроков или игрока и компьютера) состоит в том, что эти потоки должны работать одновременно.

Обязательное условие, относящееся к начальным количества раундов игроков, – они передаются через параметр командой строки в клиенте.

## Постановка задачи

Задачей данной курсовой работы является разработка игры «Три пальца», в которой должна быть возможность выбора режима игры (игрок против игрока или игрок против компьютера). Игра должны работать под управлением операционной системы семейства Unix/Linux. Игра должна быть выполнена на языке Си, а сетевая часть с помощью сокетов, представленных системной библиотекой <sys/socket.h>

## Выбор инструментальных средств

Сетевая часть программы будет реализована с помощью библиотек сокетов Беркли на Си. В нашем случае мы разрабатываем игру, поэтому нам необходимо, чтобы каждый пакет точно дошел до своего получателя в целости и сохранности, поэтому был использован протокол передачи данных семейства TCP/IP

Графический интерфейс не используется, вместо него используется консольный интерфейс.

## Выбор модели

В сетевой игре могут принимать участие только два игрока. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Фаза подготовки игры заключается в проверке сервером на правильность количеств раундов двух игроков, их сверке и произведении уже описанных в п. Введение действий, если эти количества не совпадают. В основной части игры сервер должен принимать числа пальцев, проверять их правильность, координировать ходы клиентов (игроки ходят или ждут мастера), отсылать состояния клиентов и результаты ходов.

## Выбор протокола транспортной OSI-модели

Перед тем как реализовывать сетевую часть игру, необходимо определиться, какой протокол нужно использовать - UDP или TCP/IP.

Начнем с TCP. Большим плюсом данного протокола — это его высокая надёжность, поскольку данный протокола позволяет не терять данные при передаче, запрашивая подтверждения о получении от принимающей стороны и в случае необходимости отправлять данные повторно. Но у этого протокола есть небольшой минус — это относительная низкая скорость передачи данных, за счет того, что выполнение надежной и упорядоченной передачи занимает больше времени.

Для протокола TCP есть альтернатива — протокол UDP. Данный уже протокол является противоположность по преимуществам и недостаткам TCP. Если TCP надежный, то UDP уже нет. TCP медленный, а UDP быстрый.

|  |  |  |  |
| --- | --- | --- | --- |
| Протокол TCP | | Протокол UDP | |
| Преимущества | Недостатки | Преимущества | Недостатки |
| Надежность. Подтверждения получения данных, повторной отправки в случае необходимости | Скорость. | Скорость. | Надежность. При получении отправленные данные могут приходить не полностью; |
| Упорядоченность. Гарантируется передача данных в том порядке, в котором они были отправлены | - | - | Упорядоченность. Порядок передачи данных не соблюдается |
| Метод передачи данных — потоковая. Границы фрагментов данных не имеют обозначения. | - | - | Метод передачи данных — датаграмма. Проверка пакетов на целостность осуществляется принимающей стороной только в случае получения сообщения |

Для своей курсовой работы я решил использовать TCP-протокол, поскольку мне необходимо, чтобы все пакеты всегда доходили от сервера до клиентов и от клиентов до сервера. Если до одного из двух игроков не дойдет сообщение, то произойдет нарушение очередности ходов как для второго игрока, так и для сервера, что не допустимо.

## Результат работы

В ходе выполнения данной курсовой работы удалось разработать игру «Три пальца» с двумя игровыми режимами. За обработку ведущего и каждого игрока отвечает отдельный поток.

## Заключение

Таким образом, результат курсовой работы полностью соответствует требованиям к реализации, все поставленные задачи были выполнены. Программа работает на операционных системах семейства Linux и написана на языке Си (компилятор gcc). Серверная часть программы построена с помощью протокола TCP. Созданная программа логически завершена.

# СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

Карпов В. Е, Коньков К. А. «Основы операционных систем. Курс лекций» - Учебное пособие, 2005 — 536 с.

ПРИЛОЖЕНИЕ А

# ЛИСТИНГ SERVER.C

#include "stdio.h"

#include "errno.h"

#include "unistd.h"

#include "stdlib.h"

#include "string.h"

#include "pthread.h"

#include <pthread.h>

#include "sys/types.h"

#include "sys/socket.h"

#include "sys/sem.h"

#include "sys/ipc.h"

#include "arpa/inet.h"

#include "netinet/in.h"

#include <strings.h>

#include "utility.h"

typedef struct master\_info\_struct

{

u\_int16\_t p1\_tcp\_socket\_fd;

u\_int16\_t p2\_tcp\_socket\_fd;

client\_states p1\_own\_state;

client\_states p2\_own\_state;

int p1\_score\_temp;

int p1\_score;

int p2\_score\_temp;

int p2\_score;

int round\_num;

int sem\_id;

} master\_info;

void print\_server\_port(const u\_int16\_t tcp\_socket\_fd)

{

struct sockaddr\_in socket\_address;

socklen\_t socket\_len = sizeof(socket\_address);

getsockname(tcp\_socket\_fd, (struct sockaddr\*) &socket\_address, &socket\_len);

printf("Server started in %i port\n", ntohs(socket\_address.sin\_port));

}

void print\_help\_and\_exit()

{

printf("========================= REQUIRED INPUT =========================\n");

printf("| <nothing> == To start game with any port |\n");

printf("| <port> == To start with selected port |\n");

printf("| The correct number of rounds is from 1 to 1000 |\n");

printf("==================================================================\n");

exit(0);

}

void\* p1\_thread(void\* master\_struct)

{

master\_info\* info = (master\_info\*) master\_struct;

server\_answer serv\_answ;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, PLAYER\_ONE, LOCK);

finger finger\_num;

serv\_answ = SERVER\_ERROR;

while(serv\_answ != SERVER\_OK)

{

if (read(info->p1\_tcp\_socket\_fd, &finger\_num, sizeof(finger)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, PLAYER\_ONE);

if (finger\_num <= 3 && finger\_num >= 1)

{

serv\_answ = SERVER\_OK;

if (write(info->p1\_tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, PLAYER\_ONE);

}

else //else if (finger\_num > 3 || finger\_num < 1)

{

if (write(info->p1\_tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, PLAYER\_ONE);

}

}

info->p1\_score\_temp = finger\_num;

info->p1\_own\_state = CLIENT\_STATE\_WAITING\_FOR\_MASTER;

if (write(info->p1\_tcp\_socket\_fd, &info->p1\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, PLAYER\_ONE);

step++;

sem\_set\_state(info->sem\_id, MASTER\_ONE, UNLOCK);

}

return 0;

}

void\* p2\_thread(void\* master\_struct)

{

master\_info\* info = (master\_info\*) master\_struct;

server\_answer serv\_answ;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, PLAYER\_TWO, LOCK);

finger finger\_num;

serv\_answ = SERVER\_ERROR;

while(serv\_answ != SERVER\_OK)

{

if (read(info->p2\_tcp\_socket\_fd, &finger\_num, sizeof(finger)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, PLAYER\_TWO);

if (finger\_num <= 3 && finger\_num >= 1)

{

serv\_answ = SERVER\_OK;

if (write(info->p2\_tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, PLAYER\_TWO);

}

else //else if (finger\_num > 3 || finger\_num < 1)

{

if (write(info->p2\_tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, PLAYER\_TWO);

}

}

info->p2\_score\_temp = finger\_num;

info->p2\_own\_state = CLIENT\_STATE\_WAITING\_FOR\_MASTER;

if (write(info->p2\_tcp\_socket\_fd, &info->p2\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, PLAYER\_TWO);

step++;

sem\_set\_state(info->sem\_id, MASTER\_TWO, UNLOCK);

}

return 0;

}

void\* m\_thread(void\* master\_struct)

{

master\_info\* info = (master\_info\*) master\_struct;

server\_answer serv\_answ;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, MASTER\_ONE, LOCK);

sem\_set\_state(info->sem\_id, MASTER\_TWO, LOCK);

int score\_temp = info->p1\_score\_temp + info->p2\_score\_temp;

if (score\_temp % 2 == 0)

{

printf("Player two wins the round %i!\n", step + 1);

info->p1\_score -= score\_temp;

info->p2\_score += score\_temp;

info->p2\_own\_state = CLIENT\_STATE\_ROUND\_WIN;

info->p1\_own\_state = CLIENT\_STATE\_ROUND\_LOSS;

}

else

{

printf("Player one wins the round %i!\n", step + 1);

info->p1\_score += score\_temp;

info->p2\_score -= score\_temp;

info->p1\_own\_state = CLIENT\_STATE\_ROUND\_WIN;

info->p2\_own\_state = CLIENT\_STATE\_ROUND\_LOSS;

}

if (write(info->p1\_tcp\_socket\_fd, &info->p1\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, MASTER\_ONE);

if (write(info->p2\_tcp\_socket\_fd, &info->p2\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, MASTER\_TWO);

info->p1\_score\_temp = info->p2\_score\_temp = 0;

if (step == info->round\_num - 1)

{

if (info->p2\_score > info->p1\_score)

{

printf("=========== RESULTS ===========\n");

printf("Player one wins the game!\n");

printf("===============================\n");

info->p1\_own\_state = CLIENT\_STATE\_GAME\_WIN;

info->p2\_own\_state = CLIENT\_STATE\_GAME\_LOSS;

}

else

{

printf("=========== RESULTS ===========\n");

printf("Player two wins the game!\n");

printf("===============================\n");

info->p2\_own\_state = CLIENT\_STATE\_GAME\_WIN;

info->p1\_own\_state = CLIENT\_STATE\_GAME\_LOSS;

}

}

else

info->p1\_own\_state = info->p2\_own\_state = CLIENT\_STATE\_MAKE\_TURN;

if (write(info->p1\_tcp\_socket\_fd, &info->p1\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p1\_tcp\_socket\_fd, MASTER\_ONE);

if (write(info->p2\_tcp\_socket\_fd, &info->p2\_own\_state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(info->p2\_tcp\_socket\_fd, MASTER\_TWO);

if (step == info->round\_num - 1)

break;

step++;

sem\_set\_state(info->sem\_id, PLAYER\_ONE, UNLOCK);

sem\_set\_state(info->sem\_id, PLAYER\_TWO, UNLOCK);

}

return 0;

}

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

{

//sockets setup

u\_int16\_t server\_port;

if (argc == 1)

server\_port = 0;

else if (argc == 2)

server\_port = atoi(argv[1]);

else

print\_help\_and\_exit();

uint32\_t received\_bytes = 0;

u\_int16\_t tcp\_socket\_fd;

u\_int16\_t tcp\_player\_one\_socket\_fd;

u\_int16\_t tcp\_player\_two\_socket\_fd;

socket\_address\_in server\_address;

socket\_address\_in client\_address;

bzero(&server\_address, sizeof(server\_address));

bzero(&client\_address, sizeof(client\_address));

client\_states f\_client\_state;

client\_states s\_client\_state;

if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror(NULL);

return -1;

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(server\_port);

server\_address.sin\_addr.s\_addr = htonl(INADDR\_ANY);

if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)

{

printf("Error! Can't get %i port! Tryig to get another!\n", server\_port);

server\_address.sin\_port = htons(0);

if (bind(tcp\_socket\_fd, (struct sockaddr\*) &server\_address, sizeof(server\_address)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

}

print\_server\_port(tcp\_socket\_fd);

if (listen(tcp\_socket\_fd, 1) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

tcp\_player\_one\_socket\_fd = accept(tcp\_socket\_fd, (socket\_address\*) &client\_address, &received\_bytes);

if (tcp\_player\_one\_socket\_fd < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

f\_client\_state = CLIENT\_STATE\_WAITING\_FOR\_ANOTHER\_PLAYER;

if (write(tcp\_player\_one\_socket\_fd, &f\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

printf("First player is successfully connected!\n");

tcp\_player\_two\_socket\_fd = accept(tcp\_socket\_fd, (socket\_address\*) &client\_address, &received\_bytes);

if (tcp\_player\_two\_socket\_fd < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(tcp\_player\_two\_socket\_fd, &s\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

f\_client\_state = CLIENT\_STATE\_WAITING\_FOR\_GAME\_START;

s\_client\_state = CLIENT\_STATE\_WAITING\_FOR\_GAME\_START;

if (write(tcp\_player\_one\_socket\_fd, &f\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(tcp\_player\_two\_socket\_fd, &s\_client\_state, sizeof(client\_states)) < 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

printf("Both players are successfully connected!\n");

//round num check

//need to upgrade

int p1\_round\_num, p2\_round\_num, round\_num;

if (read(tcp\_player\_one\_socket\_fd, &p1\_round\_num, sizeof(int)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (read(tcp\_player\_two\_socket\_fd, &p2\_round\_num, sizeof(int)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

server\_answer serv\_answ;

if (p1\_round\_num < 1 || p1\_round\_num > 1000)

{

serv\_answ = SERVER\_ERROR;

if (write(tcp\_player\_one\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

print\_help\_and\_exit();

}

else

{

serv\_answ = SERVER\_OK;

if (write(tcp\_player\_one\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

}

if (p2\_round\_num < 1 || p2\_round\_num > 1000)

{

serv\_answ = SERVER\_ERROR;

if (write(tcp\_player\_two\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

print\_help\_and\_exit();

}

else

{

serv\_answ = SERVER\_OK;

if (write(tcp\_player\_two\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

}

if (p1\_round\_num == p2\_round\_num)

{

round\_num = p1\_round\_num;

serv\_answ = SERVER\_OK;

if (write(tcp\_player\_two\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(tcp\_player\_one\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

}

else

{

serv\_answ = SERVER\_ERROR;

if (write(tcp\_player\_two\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(tcp\_player\_one\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

print\_help\_and\_exit();

}

//need to upgrade

//initialization section

master\_info info;

info.round\_num = round\_num;

info.p1\_tcp\_socket\_fd = tcp\_player\_one\_socket\_fd;

info.p2\_tcp\_socket\_fd = tcp\_player\_two\_socket\_fd;

info.p1\_own\_state = f\_client\_state;

info.p2\_own\_state = s\_client\_state;

info.sem\_id = semget(IPC\_PRIVATE, 4, 0600 | IPC\_CREAT);

if (info.sem\_id < 0)

{

perror("Error with semget()!\n");

print\_help\_and\_exit();

}

sem\_set\_state(info.sem\_id, PLAYER\_ONE, UNLOCK);

sem\_set\_state(info.sem\_id, PLAYER\_TWO, UNLOCK);

info.p1\_own\_state = info.p2\_own\_state = CLIENT\_STATE\_MAKE\_TURN;

if (write(info.p1\_tcp\_socket\_fd, &info.p1\_own\_state, sizeof(client\_states)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

if (write(info.p2\_tcp\_socket\_fd, &info.p2\_own\_state, sizeof(client\_states)) <= 0)

{

perror(NULL);

close(tcp\_socket\_fd);

return -1;

}

int master\_thread\_returned\_value;

pthread\_t player\_one\_thread;

pthread\_t master\_thread;

pthread\_t player\_two\_thread;

pthread\_create(&player\_one\_thread, NULL, p1\_thread, &info);

pthread\_create(&master\_thread, NULL, m\_thread, &info);

pthread\_create(&player\_two\_thread, NULL, p2\_thread, &info);

pthread\_join(master\_thread, (void\*\*) &master\_thread\_returned\_value);

printf("Master thread exit with status: %i\n", master\_thread\_returned\_value);

printf("Shutting down the server...\n");

free\_semaphores(&info.sem\_id);

return 0;

}

# ЛИСТИНГ CLIENT.C

#include "time.h"

#include "stdio.h"

#include "errno.h"

#include "unistd.h"

#include "stdlib.h"

#include "sys/sem.h"

#include "sys/ipc.h"

#include "arpa/inet.h"

#include "sys/types.h"

#include "sys/socket.h"

#include "netinet/in.h"

#include <strings.h>

#include "pthread.h"

#include <pthread.h>

#include "utility.h"

typedef struct master\_info\_struct\_client

{

int p\_score\_temp;

int p\_score;

int c\_score\_temp;

int c\_score;

int round\_num;

int sem\_id;

} master\_info\_client;

void print\_programm\_use\_and\_exit()

{

printf("Use: ./client <args>\n");

printf("Game mods options:\n");

printf("<round num> -- Player vs Computer\n");

printf("<round num> <ip-address> <port> -- Player vs Player\n\n");

printf("The correct number of rounds is from 1 to 1000\n");

exit(1);

}

void player\_versus\_player(char\* server\_ip, int server\_port, int round\_num)

{

//sockets setup

u\_int16\_t tcp\_socket\_fd = 0;

client\_states state;

socket\_address\_in server\_address;

socket\_address\_in client\_address;

bzero(&server\_address, sizeof(server\_address));

bzero(&client\_address, sizeof(client\_address));

if ((tcp\_socket\_fd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0)

{

perror(NULL);

exit(1);

}

server\_address.sin\_family = AF\_INET;

server\_address.sin\_port = htons(server\_port);

if (inet\_aton(server\_ip, &server\_address.sin\_addr) == 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (connect(tcp\_socket\_fd, (socket\_address\*) &server\_address, sizeof(server\_address)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_WAITING\_FOR\_ANOTHER\_PLAYER)

{

printf("You are successfully connected!\n");

printf("Waiting for another player...\n");

}

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_WAITING\_FOR\_GAME\_START)

{

printf("Both players are successfully connected!\n");

printf("Waiting for game start...\n");

}

//round num check

if (round\_num < 1 || round\_num > 1000)

{

printf("Wrong round number!\n");

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

}

if (write(tcp\_socket\_fd, &round\_num, sizeof(int)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

server\_answer serv\_answ;

if (read(tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (serv\_answ != SERVER\_OK)

{

printf("Wrong round number!\n");

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

}

if (read(tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (serv\_answ != SERVER\_OK)

{

printf("Check your round number and restart the game!\n");

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

}

//initialization section

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_MAKE\_TURN)

printf("You turn!\n");

//cycle of rounds

finger finger\_num;

int step = 0;

while (step < round\_num)

{

if (state == CLIENT\_STATE\_MAKE\_TURN)

{

printf("Please, write the number of fingers between 1 and 3 including!\n");

scanf("%i", (int\*) &finger\_num);

//need to upgrade on inf entering

if (write(tcp\_socket\_fd, &finger\_num, sizeof(int)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (read(tcp\_socket\_fd, &serv\_answ, sizeof(server\_answer)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (serv\_answ != SERVER\_OK)

{

printf("Wrong input\n");

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

}

//need to upgrade on inf entering

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) <= 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_WAITING\_FOR\_MASTER)

printf("Waiting for master...\n");

}

else

{

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_ROUND\_WIN)

printf("You won the round %i!\n", step + 1);

else

printf("You lost the round %i!\n", step + 1);

if (read(tcp\_socket\_fd, &state, sizeof(client\_states)) < 0)

throw\_exception\_and\_exit(tcp\_socket\_fd, CLIENT\_MAIN);

if (state == CLIENT\_STATE\_GAME\_WIN)

{

printf("You won the game!\n");

break;

}

else if (state == CLIENT\_STATE\_GAME\_LOSS)

{

printf("You lost the game!\n");

break;

}

else

printf("Its your turn!\n");

step++;

}

}

//socket close section

close(tcp\_socket\_fd);

exit(0);

}

int get\_rand\_int\_frome\_range(int min, int max)

{

if (!min)

return rand() % (max + 1);

return min + rand() + (max - min + 1);

}

void\* p\_thread(void\* master\_struct)

{

master\_info\_client\* info = (master\_info\_client\*) master\_struct;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, PLAYER\_ONE, LOCK);

finger finger\_num;

printf("Please, write the number of fingers between 1 and 3 including!\n");

scanf("%i", (int\*) &finger\_num);

//need the upgrade

if (finger\_num >= 1 && finger\_num <= 3)

printf("Success!\n");

else

{

printf("Wrong input!\n");

print\_programm\_use\_and\_exit();

}

//need the upgrade

info->p\_score\_temp = finger\_num;

step++;

sem\_set\_state(info->sem\_id, MASTER\_ONE, UNLOCK);

}

return 0;

}

void\* c\_thread(void\* master\_struct)

{

master\_info\_client\* info = (master\_info\_client\*) master\_struct;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, PLAYER\_TWO, LOCK);

finger finger\_num = get\_rand\_int\_frome\_range(1, 3);

info->c\_score\_temp = finger\_num;

step++;

sem\_set\_state(info->sem\_id, MASTER\_TWO, UNLOCK);

}

return 0;

}

void\* m\_thread(void\* master\_struct)

{

master\_info\_client\* info = (master\_info\_client\*) master\_struct;

int step = 0;

while (step < info->round\_num)

{

sem\_set\_state(info->sem\_id, MASTER\_ONE, LOCK);

sem\_set\_state(info->sem\_id, MASTER\_TWO, LOCK);

int score\_temp = info->p\_score\_temp + info->c\_score\_temp;

if (score\_temp % 2 == 0)

{

printf("Player wins the round %i!\n", step + 1);

info->p\_score -= score\_temp;

info->c\_score += score\_temp;

}

else

{

printf("Computer wins the round %i!\n", step + 1);

info->p\_score += score\_temp;

info->c\_score -= score\_temp;

}

info->p\_score\_temp = info->c\_score\_temp = 0;

if (step == info->round\_num - 1)

{

if (info->c\_score > info->p\_score)

{

printf("=========== RESULTS ===========\n");

printf("Computer wins the game!\n");

printf("===============================\n");

break;

}

else

{

printf("=========== RESULTS ===========\n");

printf("Player wins the game!\n");

printf("===============================\n");

break;

}

}

step++;

sem\_set\_state(info->sem\_id, PLAYER\_ONE, UNLOCK);

sem\_set\_state(info->sem\_id, PLAYER\_TWO, UNLOCK);

}

return 0;

}

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

{

srand(time(NULL));

int round\_num;

if (argc == 2)

{

//need the upgrade

round\_num = atoi(argv[1]);

if (round\_num >= 1 && round\_num <= 1000)

printf("Success!\n");

else

print\_programm\_use\_and\_exit();

//need the upgrade

}

else if (argc == 4)

{

int round\_num\_index\_from\_argv = 1;

char ip\_index\_from\_argv = 2;

char port\_index\_from\_argv = 3;

player\_versus\_player(argv[ip\_index\_from\_argv], atoi(argv[port\_index\_from\_argv]), atoi(argv[round\_num\_index\_from\_argv]));

}

else

print\_programm\_use\_and\_exit();

master\_info\_client info;

info.round\_num = round\_num;

info.sem\_id = semget(IPC\_PRIVATE, 4, 0600 | IPC\_CREAT);

if (info.sem\_id < 0)

{

perror("Error with semget()!\n");

print\_programm\_use\_and\_exit();

}

sem\_set\_state(info.sem\_id, PLAYER\_ONE, UNLOCK);

sem\_set\_state(info.sem\_id, PLAYER\_TWO, UNLOCK);

printf("Starting the game...\n");

int master\_thread\_returned\_value;

pthread\_t player\_one\_thread;

pthread\_t master\_thread;

pthread\_t player\_two\_thread;

pthread\_create(&player\_one\_thread, NULL, p\_thread, &info);

pthread\_create(&master\_thread, NULL, m\_thread, &info);

pthread\_create(&player\_two\_thread, NULL, c\_thread, &info);

pthread\_join(master\_thread, (void\*\*) &master\_thread\_returned\_value);

printf("Master thread exit with status: %i\n", master\_thread\_returned\_value);

printf("Shutting down the client...\n");

free\_semaphores(&info.sem\_id);

return 0;

}

# ЛИСТИНГ UTILITY.H

#ifndef UTILITY\_H

#define UTILITY\_H

#include "stdio.h"

#include "stdlib.h"

#include "unistd.h"

#include "sys/sem.h"

#define UNLOCK 1

#define LOCK -1

#define PLAYER\_ONE 0

#define PLAYER\_TWO 1

#define MASTER\_ONE 2

#define MASTER\_TWO 3

#define CLIENT\_MAIN 4

typedef struct sockaddr socket\_address;

typedef struct sockaddr\_in socket\_address\_in;

typedef struct sockaddr socket\_address;

typedef struct sockaddr\_in socket\_address\_in;

typedef int finger;

typedef enum server\_answer\_enum

{

SERVER\_OK,

SERVER\_ERROR

} server\_answer;

typedef enum client\_states\_enum

{

CLIENT\_STATE\_WAITING\_FOR\_ANOTHER\_PLAYER,

CLIENT\_STATE\_WAITING\_FOR\_GAME\_START,

CLIENT\_STATE\_MAKE\_TURN,

CLIENT\_STATE\_WAITING\_FOR\_MASTER,

CLIENT\_STATE\_ROUND\_WIN,

CLIENT\_STATE\_ROUND\_LOSS,

CLIENT\_STATE\_GAME\_WIN,

CLIENT\_STATE\_GAME\_LOSS

} client\_states;

void sem\_set\_state(int sem\_id, int num, int state);

void throw\_exception\_and\_exit(u\_int16\_t tcp\_socker\_fd, char thread\_num);

void free\_semaphores(int\* sem\_id);

#endif

# ЛИСТИНГ UTILITY.C

#include "utility.h"

void sem\_set\_state(int sem\_id, int num, int state)

{

struct sembuf op;

op.sem\_op = state;

op.sem\_flg = 0;

op.sem\_num = num;

semop(sem\_id, &op, 1);

}

void throw\_exception\_and\_exit(u\_int16\_t tcp\_socker\_fd, char thread\_num)

{

printf("Can't read or send data. Thread number: %i. Shutting down the server...\n", thread\_num);

perror(NULL);

close(tcp\_socker\_fd);

exit(1);

}

void free\_semaphores(int\* sem\_id)

{

char resource\_delete\_command\_buff[124];

sprintf(resource\_delete\_command\_buff, "ipcrm -s %i", \*sem\_id);

system(resource\_delete\_command\_buff);

\*sem\_id = 0;

}