Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Информационных технологий и систем»

**КУРСОВАЯ РАБОТА**

по дисциплине: «Операционные системы»

на тему: «Многопользовательская сетевая игра: «3 Пальца».

Разработал:

Студент группы 8091

Бочкарев Б. А.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Проверил:

Ананьев В. В.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2020г.

Великий Новгород

2020

Содержание

[Цель работы 3](#_Toc42234369)

[Разработка проекта 3](#_Toc42234370)

[1.1 Задание 3](#_Toc42234371)

[1.2 Выбор модели 3](#_Toc42234372)

[1.3 Выбор протокола транспортного уровня OSI-модели 4](#_Toc42234373)

[1.4 Алгоритм решения задачи 5](#_Toc42234374)

[1.5 UML-диаграмма взаимодействия потоков 6](#_Toc42234375)

[Реализация проекта 6](#_Toc42234376)

[2.1 Структура клиента и сервера 6](#_Toc42234377)

[2.2 Программный интерфейс сервера/клиента 6](#_Toc42234378)

[2.3 Системные вызовы 7](#_Toc42234379)

[2.4 Пример работы приложения 9](#_Toc42234380)

[2.5 Результат работы 11](#_Toc42234381)

[Вывод 11](#_Toc42234382)

[Список используемой литературы 11](#_Toc42234383)

# **Цель работы**

Целью данной работы является создание консольной игры с 2 игроками. Управлять каждым игроком может как человек, так и компьютер. Игра должна быть написана на языке C или C++ под OS Linux.

Суть игры: игроки одновременно показывают 1, 2 или 3 пальца. Далее ведущий подсчитывает сумму очков S (по общему количеству показанных пальцев). Если сумма четная, то выигрывает второй игрок, иначе выигрывает первый игрок. Выигравший игрок добавляет эту сумму S к своим очкам, проигравший вычитает S из своих очков. Изначально у каждого игрока ноль очков.

Количество раундов задается в параметрах командной строки при запуске приложения. После окончания игры требуется вывести финальное количество очков каждого игрока.

# **Разработка проекта**

* 1. Задание

Заданием к курсовой работе я выбрал написание игры на языке C или C++ под OS Linux. Количество раундов задается в параметрах командной строки при запуске приложения. Созданная мной игра называется «3 пальца».

* 1. Выбор модели

Игра написана с использованием модели клиент-сервер. В игре принимает участие 2 человека, и 1 ведущий. В моем решении роль ведущего выполняет сервер. Клиент устроен довольно просто: при запуске клиента нужно указать, клиент является ботом или человеком.

В самом клиенте выполняется создание потока, отправка на сервер номер игрока и ввод количества пальцев, которые впоследствии отправляются на сервер для подсчета и обработки. При запуске сервера нужно указать в качестве аргумента количество раундов. Сам сервер принимает количество пальцев каждого игрока, подсчитывает сумму, определяет победителя этого раунда и отправляет результат игрокам. Когда количество раундов подходит к концу сервер отправляет сообщение с номером игрока победителя.

Протокол взаимодействия получается таким: 2 клиента отправляют кол-во пальцев, сервер принимает эти данные, суммирует, выбирает победителя раунда и отправляет клиентам результат раунда, если раунд последний, то сервер также выбирает победителя и отправляет сообщение с поздравлением, и завершает работу. Схематично алгоритм показан на рисунке 1.

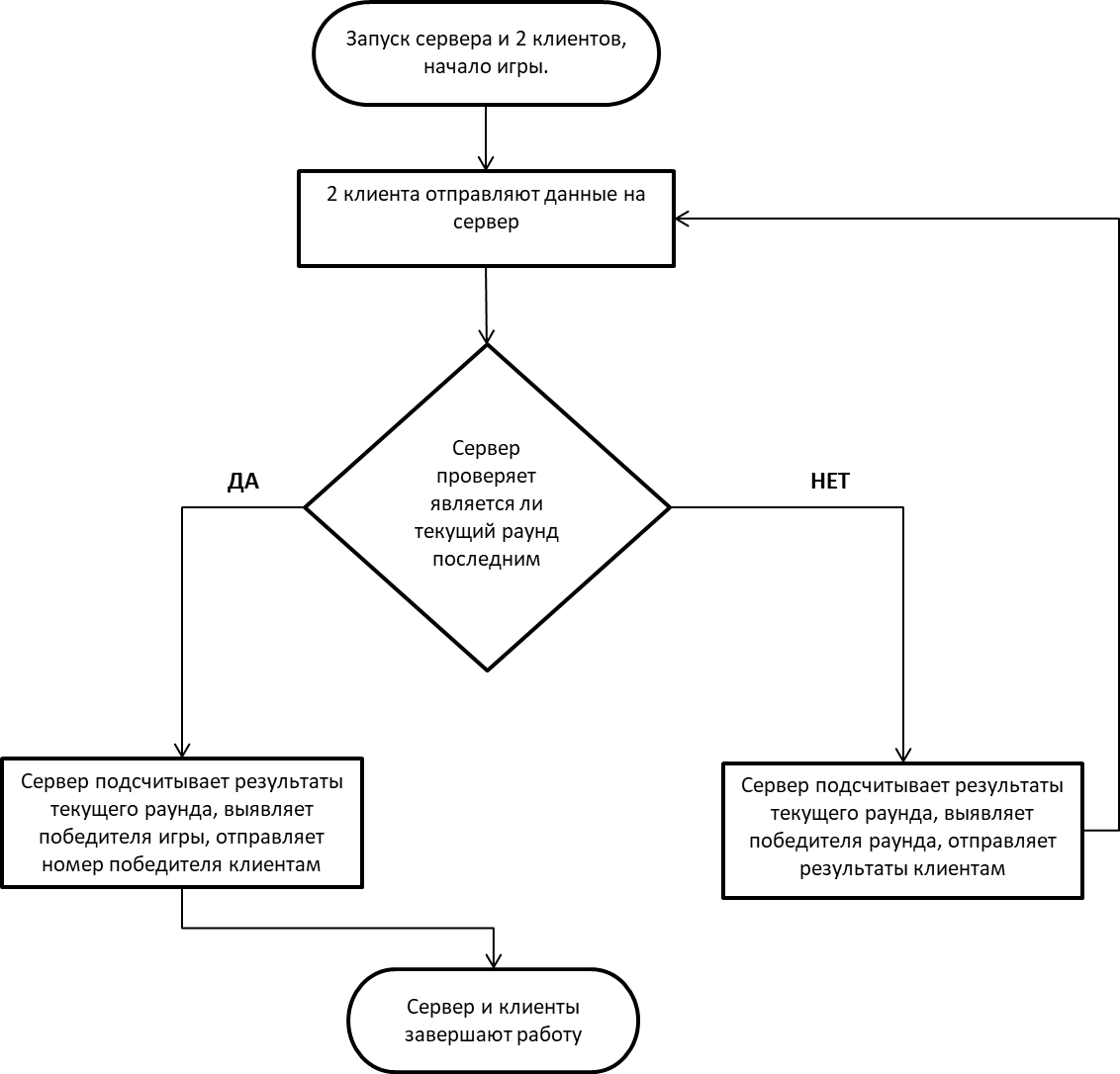


Рисунок 1. – Алгоритм протокола взаимодействия

* 1. Выбор протокола транспортного уровня OSI-модели

Рассмотри оба протокола:

TCP — это протокол, использующий принцип надежного соединения. Это означает, что вы устанавливаете соединение между двумя компьютерами, и затем пересылаете данные между ними подобно тому, как если бы вы записывали информацию в файл на одном компьютере, а на другом — считывали бы ее из того же файла. При этом соединение считается надежным и последовательным — то есть, вся информация, которую вы посылаете, гарантированно должна дойти до получателя в том же порядке, в каком была отправлена. Также TCP соединение можно считать непрерывным потоком данных — протокол сам заботится о разбивке данных на пакеты и пересылке их по сети.

UDP использует простую модель передачи, без неявных «рукопожатий» для обеспечения надёжности, упорядочивания или целостности данных. Таким образом, UDP предоставляет ненадёжный сервис, и датаграммы могут прийти не по порядку, дублироваться или вовсе исчезнуть без следа. UDP подразумевает, что проверка ошибок и исправление либо не нужны, либо должны исполняться в приложении. Чувствительные ко времени приложения часто используют UDP, так как предпочтительнее сбросить пакеты, чем ждать задержавшиеся пакеты.

Так как я выбрал игру, в которой не важна скорость взаимодействия в реальном времени в связи с тем, что игроки отправляют значение и ждут результат, мой выбор пал на TCP.

* 1. Алгоритм решения задачи

Алгоритм решения задачи следующий: написать клиентскую часть, в которой реализовать создание отдельного потока для каждого игрока, возможность игры с компьютером, ввод и отправку количества пальцев. Написать серверную часть, в которой будет происходить подсчет суммы, отправка результатов и выбор победителя.

Схематично данный алгоритм отражён на рисунке 2.

Рисунок 2. – Алгоритм написания игры «3 пальца»

* 1. UML-диаграмма взаимодействия потоков

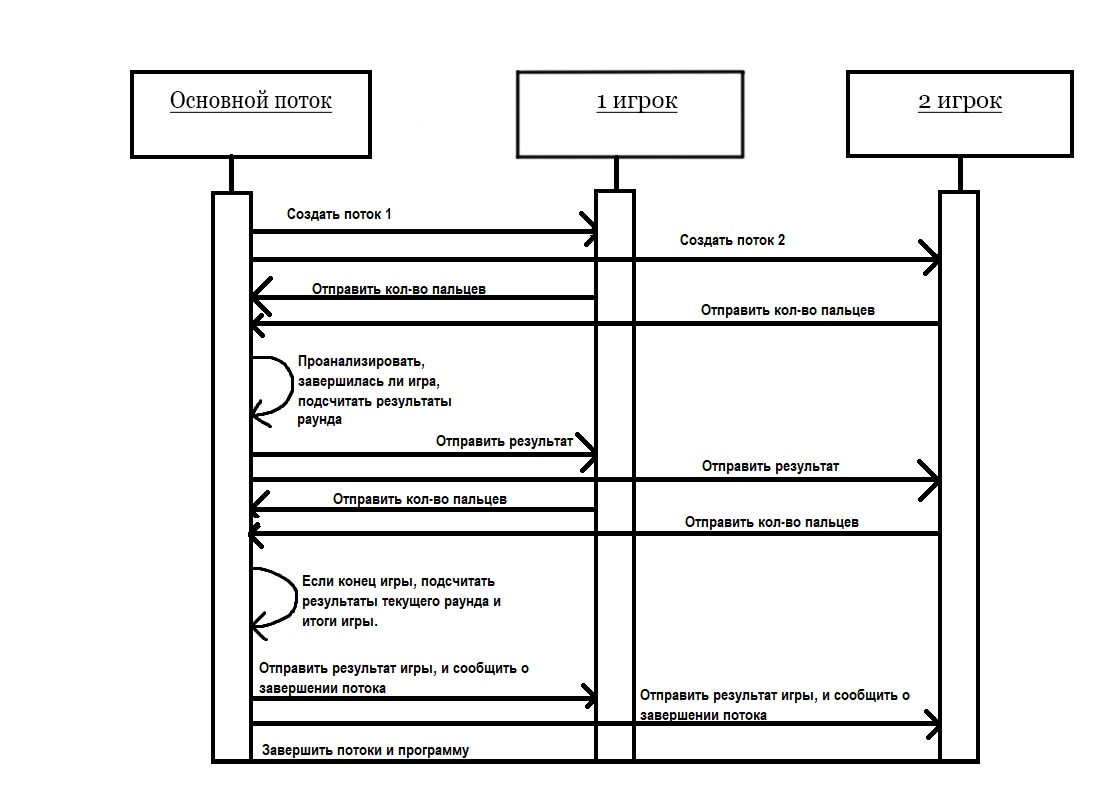


Рисунок 3. - UML-диаграмма взаимодействия потоков

# **Реализация проекта**

* 1. Структура клиента и сервера

Сервер находится в файле server.cpp, клиент в файле client.cpp. Также в работе есть Makefile и readme.

* 1. Программный интерфейс сервера/клиента

Для начала использования введите команду *make.*

После этого запустите сервер следующей командой: *./server port n*, где n – количество раундов игры.

После запуска сервера мы можем подключить игроков (в выбранной игре 2-их). Для этого вводим команду: *./client ip:port m*, где m – Игрок-человек, если ввести 0; игрок-компьютер, если ввести 1. При запуске клиента ему приходит сообщение с его номером.

* 1. Системные вызовы

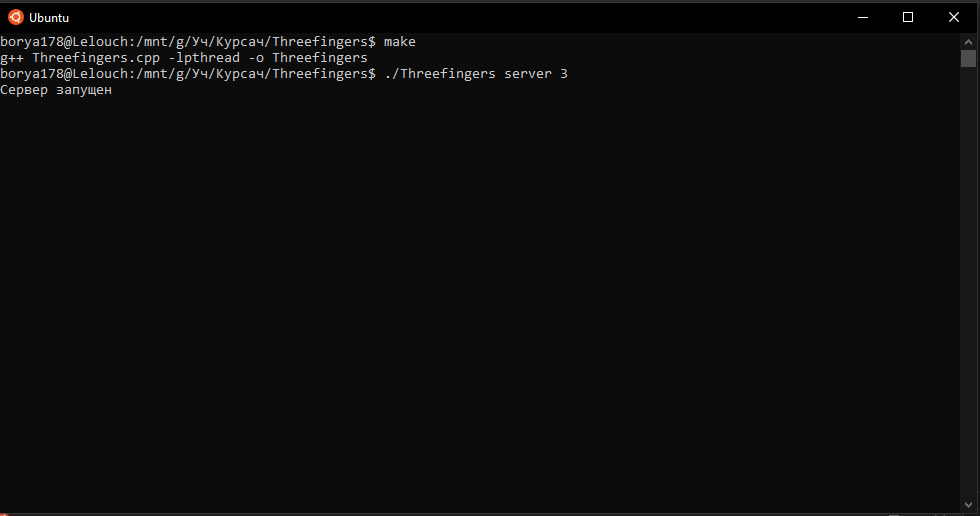
Основные системные вызовы отражены в таблице 1.

Таблица 1. – Системные вызовы

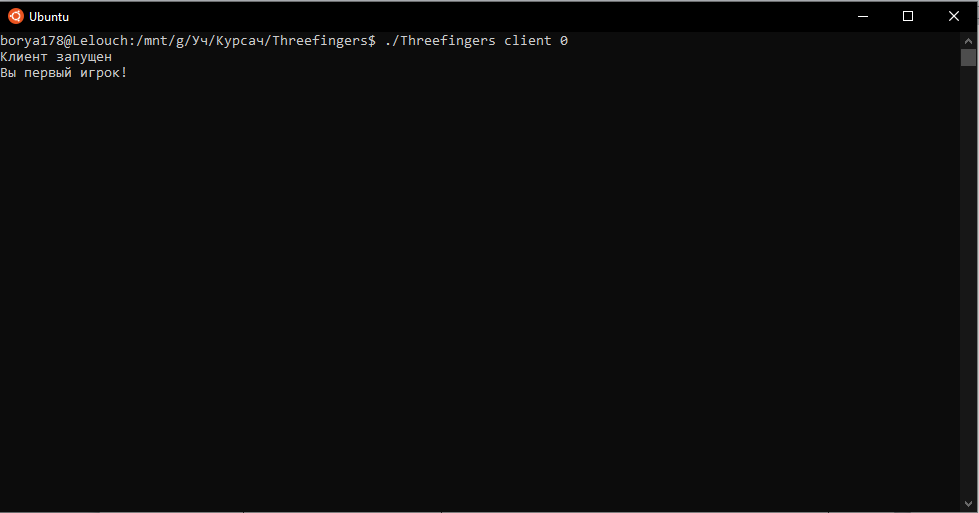
| Системный вызов | Описание |
| --- | --- |
| send() | Функция служит для записи данных в сокет. Первый аргумент - сокет-дескриптор, в который записываются данные. Второй и третий аргументы - соответственно, адрес и длина буфера с записываемыми данными. Четвертый параметр - это комбинация битовых флагов, управляющих режимами записи. |
| close() | Эта функция закрывает сокет и разрывает все соединения с этим сокетом. В отличие от функции shutdown функция close.может дожидаться окончания всех операций с сокетом, обеспечивая «нормальное», а не аварийное закрытие соединений. |
| exit() | Функция exit выполняет немедленное завершение программы. |
| recv() | Функция служит для чтения данных из сокета.  Первый аргумент - сокет-дескриптор, из которого читаются данные. Второй и третий аргументы - соответственно, адрес и длина буфера для записи читаемых данных. Четвертый параметр - это комбинация битовых флагов, управляющих режимами чтения. |
| bind() | Эта функция используется сервером для присваивания сокету имени. До выполнения функции bind (т.е. присваивания какого-либо имени, вид которого зависит от адресного домена) сокет недоступен программам-клиентам. |
| listen() | Функция listen используется сервером, чтобы информировать ОС, что он ожидает («слушает») запросы связи на данном сокете. Без такой функции всякое требование связи с этим сокетом будет отвергнуто. |
| accept() | Эта функция используется сервером для принятия связи на сокет. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то функция accept возвращает новый сокет-дескриптор, через который и происходит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция accept блокирует другие запросы связи с данным сервером, а после установления связи «прослушивание» запросов возобновляется. |
| socket() | Эта функция используется для создания сокета. Первый параметр - домен - накладывает определенные ограничения на формат используемых процессом адресов и их интерпретацию. Второй параметр определяет тип канала связи с сокетом, который должен быть использован. (SOCK\_STREAM - при этом типе связи поступающим в канал байтам информации гарантируется «доставка» в порядке их поступления; пока непрерывный поток байтов не прекратится, никакие другие данные приниматься каналом не будут (аналогом такой связи является pipe-механизм)) Третий параметр позволяет программисту выбрать нужный протокол для канала связи. Если этот параметр равен нулю, ОС выберет нужный протокол автоматически. |
| connect() | Функция connect используется процессом-клиентом для установления связи с сервером. Первый аргумент - сокет-дескриптор клиента. Второй аргумент - указатель на адрес сервера (структура sockaddr) для соответствующего домена. Третий аргумент - целое число - длина структуры адреса. Функция возвращает 0, если вызов успешный, и -1 иначе. |

* 1. Пример работы приложения

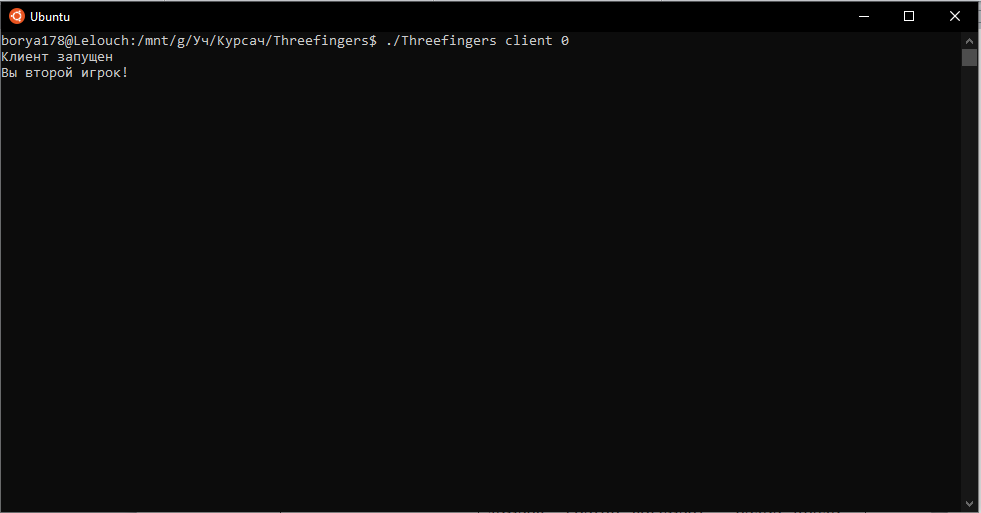
Запуск сервера с 3 раундами



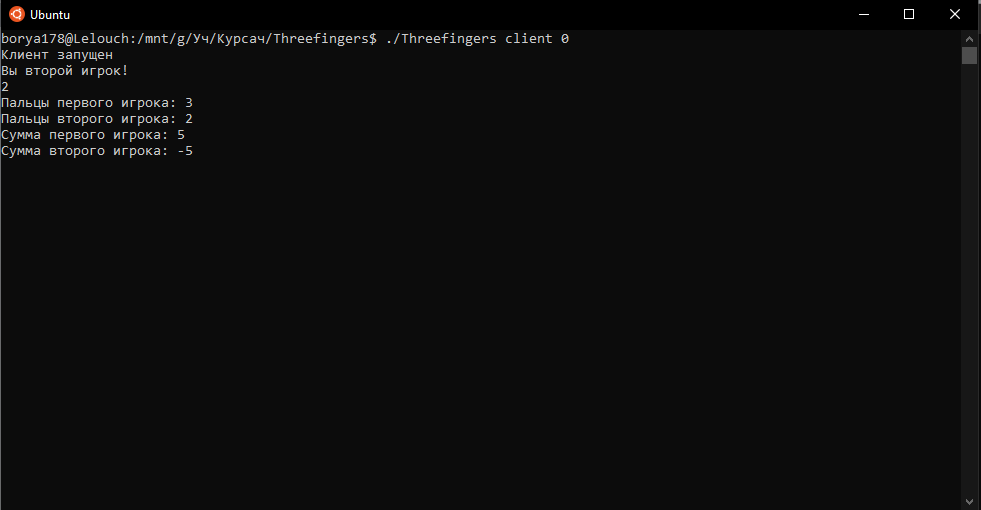
Запуск первого игрока-человека



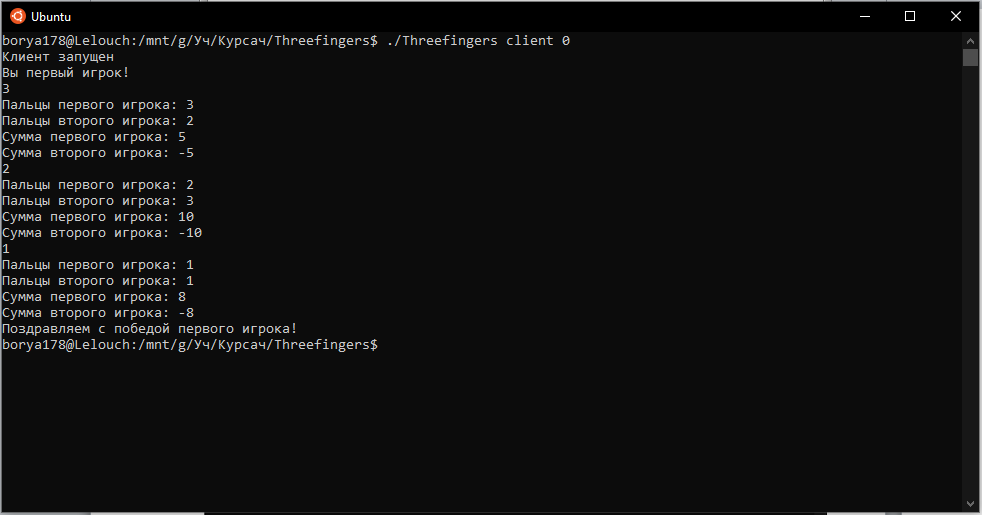
Запуск второго игрока-человека



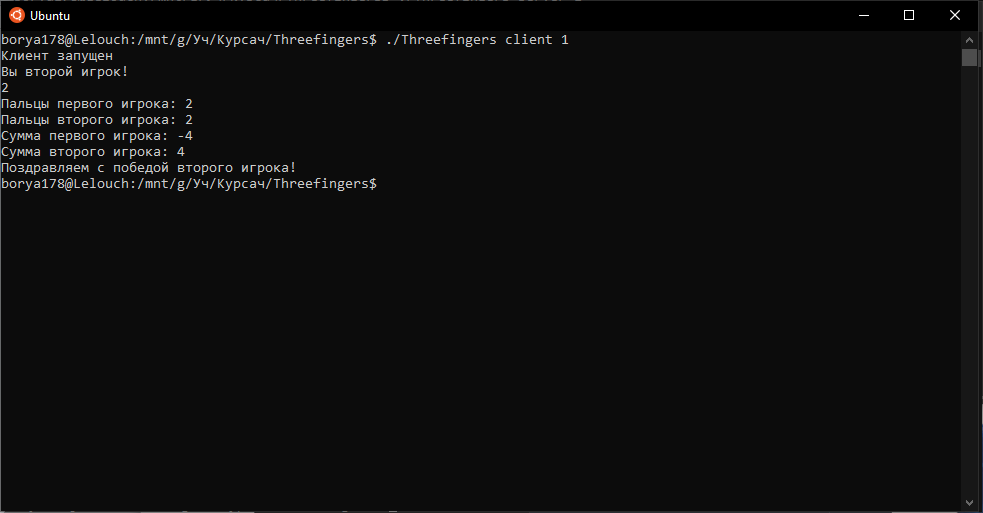
После ввода кол-ва пальцев (обоими игроками)



Полностью сыгранная игра



Запуск игрока-бота



* 1. Результат работы

В итоге курсовой работы была написана консольная игра (код программы представлен в приложение А), в которой можно задать количество раундов и выбрать с каким противником будешь играть, с человеком или компьютером. Также в данной игре можно заставить 2 компьютера играть друг против друга.

# **Вывод**

В результате работы я лучше разобрался в создании программ со структурой клиент-сервер, стал больше понимать, как устроено сетевое соединение в онлайн играх, мне стали более понятны протоколы TCP/UDP. Задание полностью выполнено, соответствует всем требованиям.

# **Список используемой литературы**

1. Карпов В.Е., Коньков К.А. - Основы операционных систем. Практикум
2. Glenn Fiedler. Сетевое программирование для разработчиков игр. Часть 1: UDP vs. TCP [электронный ресурс] // Хабр. – Режим доступа: <https://habr.com/ru/post/209144/> Дата обращения 01.06.20
3. Glenn Fiedler. Сетевое программирование для разработчиков игр. Часть 2: UDP vs. TCP [электронный ресурс] // Хабр. – Режим доступа: <https://habr.com/ru/post/209524/> Дата обращения 01.06.20
4. Интерактивная система просмотра системных руководств (man-ов) [электронный ресурс] // OpenNET. – Режим доступа: <https://www.opennet.ru/man.shtml>

**Приложение А**

**Код программы**

**server.cpp**

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include <clocale>

using namespace std;

#define MAX\_CLIENTS 2

int fingers[2];

int scores[2];

int roundCounter = 0;

bool poor\_player[2] = {false, false};

struct Client

{

int socket;

int id;

int round\_count;

pthread\_t thread;

};

Client \*clients[MAX\_CLIENTS] = {NULL};

int findSocket() //поиск свободного сокета

{

for (size\_t i = 0; i < MAX\_CLIENTS; ++i)

{

if (clients[i] == NULL)

{

return i;

}

}

}

void \*client\_receive(void \*data) //Сервер отправляет

{

Client \*player = (Client \*)data;

send(clients[player->id]->socket, &player->id, sizeof(player->id), 0); //отправляем номер игрока клиенту

int summa;

int round\_count = player->round\_count;

while (true)

{

int buffer;

char message[70];

if (roundCounter == round\_count) //проверка на окончание игры

{

if (scores[0] > scores[1])

{

strcpy(message, "Поздравляем с победой первого игрока!");

}

else if (scores[1] > scores[0])

{

strcpy(message, "Поздравляем с победой второго игрока!");

}

else if (scores[0] = scores[1])

{

strcpy(message, "Ничья");

}

for (int i = 0; i < MAX\_CLIENTS; i++)

{

char status = 2;

if (clients[i] != NULL)

{

send(clients[i]->socket, &status, sizeof(status), 0); //отправляем статус = завершение игры

send(clients[i]->socket, message, sizeof(message), 0); //отправляем сообщение с победителем

}

}

close(clients[0]->socket); //закрывает соединение с сокетом

close(clients[1]->socket); //закрывает соединение с сокетом

exit(0); //выход

}

int n;

do

{

n = recv(player->socket, &buffer, sizeof(int), 0);

} while (poor\_player[player->id] && n != 0);

poor\_player[player->id] = true;

if (n == 0)

{

return 0;

}

//cout << "Полученно сообщение от клиента № " << player->id << ": " << buffer << endl;

char status = 1;

if (player->id == 0)

{

fingers[0] = buffer;

strcpy(message, "Первый игрок сделал ход и ждёт вас!");

if (!poor\_player[1])

{

send(clients[1]->socket, &status, sizeof(status), 0);

send(clients[1]->socket, message, sizeof(message), 0);

}

}

else if (player->id == 1)

{

fingers[1] = buffer;

strcpy(message, "Второй игрок сделал ход и ждёт вас!");

if (!poor\_player[0])

{

send(clients[0]->socket, &status, sizeof(status), 0);

send(clients[0]->socket, message, sizeof(message), 0);

}

}

if (fingers[0] != 0 && fingers[1] != 0)

{

poor\_player[0] = false;

poor\_player[1] = false;

summa = fingers[0] + fingers[1];

if (summa % 2 == 0)

{

scores[1] += summa;

scores[0] -= summa;

if (scores[0] < 0)

{

scores[0] = 0;

}

}

else

{

scores[0] += summa;

scores[1] -= summa;

if (scores[1] < 0)

{

scores[1] = 0;

}

}

for (int i = 0; i < MAX\_CLIENTS; i++)

{

char status = 0;

if (clients[i] != NULL)

{

send(clients[i]->socket, &status, sizeof(status), 0);

send(clients[i]->socket, scores, sizeof(scores), 0);

send(clients[i]->socket, fingers, sizeof(fingers), 0);

}

}

memset(fingers, 0, sizeof(fingers));

roundCounter++;

}

}

}

void server(int port, int n)

{

for (size\_t i = 0; i < MAX\_CLIENTS; ++i)

{

clients[i] = NULL;

}

struct sockaddr\_in addr;

int listener = socket(AF\_INET, SOCK\_STREAM, 0); //В аргументах, первый параметр домен, второй канал связи, третий протокол для канала связи

if (listener < 0)

{

perror("Ошибка сокета");

return;

}

addr.sin\_family = AF\_INET; //Internet-домен

addr.sin\_port = htons(port); //Порт, Функция htonl() преобразует узловой порядок расположения байтов положительного целого hostlong в сетевой порядок расположения байтов

addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); //INADDR\_ANY - это "any address"

if (bind(listener, (struct sockaddr \*)&addr, sizeof(addr)) < 0) //Связать сокет с IP-адресом и портом

{

perror("Ошибка бинда");

return;

}

listen(listener, 1); //Объявить о желании принимать соединения. Слушает порт и ждет когда будет установлено соединение

while (true)

{

int sock = accept(listener, NULL, NULL); //Принять запрос на установку соединения

if (sock < 0)

{

perror("Accept error");

}

else //создаем нового клиента

{

Client \*user = new Client;

user->socket = sock;

int id = findSocket();

user->id = id;

user->round\_count = n;

clients[id] = user;

pthread\_create(&user->thread, NULL, client\_receive, user);

}

}

}

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

{

setlocale(LC\_ALL, "Russian");

if (argc < 3)

{

cout << "Не указан аргумент" << endl;

return 1;

}

int port = atoi(argv[1]);

int n = atoi(argv[2]);

if (n <= 0)

{

cout << "Раундов не может быть меньше 1!" << endl;

return 1;

}

cout << "Сервер запущен" << endl;

server(port, n);

}

**client.cpp**

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <iostream>

#include <clocale>

#include <ctime>

using namespace std;

int fingers[2];

int scores[2];

void \*server\_receive(void \*data) //клиент отправляет

{

int sock = \*((int \*)data);

char message[70];

char status;

int id;

memset(message, 0, sizeof(message));

recv(sock, &id, sizeof(id), 0); //определяем какой игрок

if (id == 0)

{

cout << "Вы первый игрок!" << endl;

}

else

{

cout << "Вы второй игрок!" << endl;

}

while (true)

{

recv(sock, &status, sizeof(status), 0);

if (status == 0)

{

recv(sock, scores, sizeof(scores), 0);

recv(sock, fingers, sizeof(fingers), 0);

if (fingers[0] != 0 && fingers[1] != 0)

{

cout << "Пальцы первого игрока: " << fingers[0] << endl;

cout << "Пальцы второго игрока: " << fingers[1] << endl;

cout << "Сумма первого игрока: " << scores[0] << endl;

cout << "Сумма второго игрока: " << scores[1] << endl;

memset(fingers, 0, sizeof(fingers));

}

}

else

{

recv(sock, message, sizeof(message), 0);

cout << message << endl;

if (status == 2)

{

close(sock);

exit(0);

}

}

}

}

void client(char \*ip\_port, int isBot)

{

char \*ip = strchr(ip\_port, ':');

int port = atoi(ip + 1);

ip[0] = 0;

struct sockaddr\_in addr;

int sock = socket(AF\_INET, SOCK\_STREAM, 0);

if (sock < 0)

{

perror("Ошибка сокета");

return;

}

addr.sin\_family = AF\_INET; //Internet-домен

addr.sin\_port = htons(port); // Порт

addr.sin\_addr.s\_addr = inet\_addr(ip\_port);

cout << ip\_port << endl;

cout << port << endl; //ip

if (connect(sock, (struct sockaddr \*)&addr, sizeof(addr)) < 0) // Установить соединение

{

perror("Ошибка подключения");

return;

}

else if (isBot)

{

pthread\_t thread; //идентификатор потока

pthread\_create(&thread, NULL, server\_receive, &sock); //создаем поток

sleep(1); //задержка перед отправкой сообщения

while (true)

{

int buffer = rand() % 3 + 1;

send(sock, &buffer, sizeof(buffer), 0);

cout << buffer << endl;

sleep(1);

}

}

else

{

pthread\_t thread;

pthread\_create(&thread, NULL, server\_receive, &sock);

while (true)

{

int buffer;

cin >> buffer;

if (buffer < 1 || buffer > 3)

{

cout << "Вы не можете ввести значение меньше 1 и больше 3, пожалуйста введит снова! " << endl;

}

else

{

send(sock, &buffer, sizeof(buffer), 0); //отправляем сообщение на сервер с нашим значением

}

}

}

}

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

{

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

if (argc < 3)

{

cout << "Не указан аргумент" << endl;

return 1;

}

char \*port = argv[1];

int isBot = atoi(argv[2]);

if (isBot < 0 || isBot > 1)

{

cout << "Неправильный аргумент, для игры с человеком введите 0, для игры с ботом 1" << endl;

return 1;

}

cout << "Клиент запущен" << endl;

client(port, isBot);

}

**Makefile**

all:

g++ server.cpp -lpthread -o server

g++ client.cpp -lpthread -o client