Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**Сетевая игра «Шашки»**

Курсовая работа по дисциплине:

«Операционные системы»

Выполнила

Студент группы №9091

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Баринов Д. А.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021

Проверил

Преподаватель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Ананьев. В. В.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021

**Великий Новгород**

**2021**

**ОГЛАВЛЕНИЕ**

**ЦЕЛЬ РАБОТЫ**...........................................................................................................................3

**ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА**……………….………………………………..……………………………………..4

1.1 Постановка задач………………………………………………………………….……………..…………....4

1.2 Создание модели работы сервера…………………………………………………………………….4

1.3 Создание модели работы клиента…………………………………………………………………….5

1.4 Выбор протокола транспортного уровня………………………………….……………………….6

1.5 Протокол взаимодействия сервера с клиентом………………………………………………..6

1.6 Решение Блокировки ввода/вывода………………………………….................................7

**ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА**…………………………………………………….....................................

2.1 Интерфейс сервера……………………………………………………………………………………………..

2.2 Список основных функций клиента и сервера…………………………………………………..

2.3 Список системных вызовов операционной системы…………………………………………

2.4 Пример работы приложения……………………………………………………………………………..

**РЕЗУЛЬТАТ РАБОТЫ**.................................................................................................................

**ВЫВОД**………………...………………………………………………………………………………………………………………

**ПРИЛОЖЕНИЕ А**…………………………………………………………………………………………………………………..

ЛИСТИНГ А1. Файл server.cpp…..………...………………………………….……………………………...

ЛИСТИНГ А2. Файл user.cpp…………………………………………………………………………………….

**ЦЕЛЬ РАБОТЫ**

Целью курсовой работы заключается в получение практических навыков в создание многопользовательских сетевых приложений под управлением операционной системы семейства Unix.

**Глава 1**

**РАЗРАБОТКА ПРОЕКТА**

*1.1 Постановка задач*

Целью данной работы является создание сетевой игры «Шашки».

**Описание:**

количество игроков = 2

на поле 8x8 расположены шашки и шахматном порядке. По 12 на каждого игрока снизу и сверху.

При ходе каждый игрок выбирает координаты своей шашки (x, y) и выбирает ход из не более двух вариантов:

ход = {(x-1, y+1), (x+1, y+1)}, если игрок управляет нижними шашками.

ход = {(x-1, y-1), (x+1, y-1)}, если за верхних.

также ход не может быть выполнен, если он приведёт к выходу за границы карты.

при достижении противоположного конца доски, шашка превращается в дамку

ход = { (x+n, y+n), (x+n, y-n) | n ∈ Z )}.

Данное приложение должно работать под управлением операционной системы семейства Unix/Linux. Сервер и клиент должны быть выполнены на языке программирования C++ с использованием сокетов, предоставляемых системной библиотекой <sys/socket.h>. Клиент и сервер должны быть как два отдельных процесса. Клиент должен иметь графический интерфейс и инструменты ввода/вывода, а сервер должен содержать саму логику игры.

* 1. *Модель работы сервера*

ЗАПУСК СЕРВЕРА;

ОЖИДАНИЕ ИГРОКА 1;

ОЖИДАНИЕ ИГРОКА 2;

ЗАПУСК ИГРЫ:

ПОКА НЕТ ПОБЕДИТЕЛЯ: {

ОЖИДАНИЕ ВВОДА ПЕРВОГО ИГРОКА;

ОБРАБОТКА ХОДА;

ОЖИДАНИЕ ВВОДА ВТОРОГО ИГРОКА;

ОБРАБОТКА ХОДА;

}

ОБРАБОТКА РЕЗУЛЬТАТОВ;

ОТПРАВКА РЕЗУЛЬТАТОВ;

ЗАВЕРШЕНИЕ РАЮБТЫ;

* 1. *Модель работы пользователя*

ЗАПУСК ПОЛЬЗОВАТЕЛЯ;

ОЖИДАНИЕ ПОЯВЛЕНИЯ СЕРВЕРА;

ПОДКЛЮЧЕНИЕ;

ОЖИДАНИЕ ПОДКЛЮЧЕНИЯ ДРУГОГО ИГРОКА;

НАЧАЛО ИГРЫ;

ПОКА НЕТ ПОБЕДИТЕЛЯ: {

ОЖИДАНИЕ ХОДА;

СДЕЛАТЬ ХОД;

ЕСЛИ ХОД НЕ ПРАВИЛЬНЫЙ: {

СДЕЛАТЬ ХОД ЕЩЁ РАЗ;

}

}

ЗАВЕРШИТЬ РАБОТУ;

*1.4 Выбор протокола транспортного уровня*

Для того, чтобы сервер и клиент могли обмениваться информацией, необходимо сначала выбрать по какому из протоколов TCP/UDP будет происходить обмен. Необходимо выяснить их основные различия.

**TCP** - это надежная служба доставки потока, гарантирующая доставку потока данных из одного хоста в другой без дублирования и потерь данных. Так как передача пакетов не является надежной, то применяется метод подтверждения приема с повторной передачей, гарантирующий надежную доставку пакетов. Для этого базового метода требуется, чтобы приемник отправлял сообщение с подтверждением при приеме данных.

**UDP** – протокол, который обеспечивает быструю отправку данных. Данный протокол имеет ряд недостатков. Сообщения поступают в любом порядке, и то, которое отправлено первым, может быть получено последним. Доставка сообщений в UDP не гарантируется. Сообщения могут потеряться, или могут быть получены две копии одного и того же сообщения.

В моём случае необходимо использовать протокол TCP так как клиенту

важно, чтобы его данные дошли до сервера в первозданном виде. Если игрок выберет один ход, а сервер получит другой, то это будет не честно.

*1.5 Протокол взаимодействия клиента и сервера*

Взаимодействия между клиентом и сервером осуществляется через обмен сообщениями, которые представляют из себя структуру. Иногда обмен данными происходит не целой структурой, а небольшими переменными. Обмен сообщениями происходит следующим образом.

В начале работы приложения серверу необходимо знать сокеты клиентов, чтобы общаться с ними.

После подключения начинается сама игра, т. е. обмен сообщениями межу пользователем и сервером.

Когда начинается ход пользователя, то он вводит две строки. Первая – это координаты шашки, которую он хочет выбрать. Вторая – координаты той клетки, на которую он хочет переместить выбранную шашку.

После чего клиент принимает сообщение от сервера с подтверждение, или сообщение о неправильном ходе.

1. message – структура, которая является основным способом общения пользователя и сервера:

|  |  |
| --- | --- |
| **Имя переменной** | **Размер, байт** |
| Word\_1 | 5 |
| Word\_2 | 6 |

*1.6 Решение проблемы блокировки при вводе выводе*

Обычно приём данных с сокета выполняется в блокирующем режиме. В случае, когда данных на сокете нету, то поток, в котором происходит чтение, блокируется до их появления. Подобная работы представляет проблему для приложения, которое должно проверять данные, которые пришли к нему, без торможения графического интерфейса. Помимо этого, проблемы возникают, когда отправляются данные на сервер. Во время боя игроки ходят параллельно. Таким образом игра наделена следующими свойствами:

**Многопоточный режим.** Данное свойство даёт возможность сохранить параллельность ходов. Когда данные приходят на сервер, то заранее проверяется состояние клиента, который отправил данные.

**Чтение данных из сокета в неблокирующем режиме.** Обычно чтение данных с сокета происходит в блокирующем режиме, что создаёт проблемы для графического интерфейса приложения. Приложение должно постоянно рисовать сцену для отображения. Если приложение начнёт считывать данные с сокета, то оно остановиться на этом моменте, пока все данные не придут. А значит приложение не будет отображать карту, то есть остановится на одном моменте и прекратить вывод в консоль. Данное свойство помогает постоянно отображать графический интерфейс и читать данные, которые были отправлены сервером без затормаживания графического интерфейса.

**Глава 2**

**РЕАЛИЗАЦИЯ ПРОЕКТА**

*2.1 Программный интерфейс сервера*

В процессе своей работы клиент формирует сообщения и отправляет его на сервер.

Сервер принимает это сообщение и обрабатывает его на наличие ошибок.

Если ошибок не найдено, то сервер записывает новое состояние шашки на карте, отправляет клиенту сообщение о том, что его ход был сделан, а в след отправляет обновлённый экран, с новым положение шашек.

Если же ошибки были обнаружены (неправильная строка, выход за пределы карты или ход, который нельзя делать по правилам, то сервер отправляет клиенту сообщение об ошибке.

*2.2 Список основных функций клиента и сервера.*

**Класс Board**

|  |  |
| --- | --- |
| **Название метода** | **Описание** |
| bool Board::move\_by\_rule(Cord pos\_1, Cord pos\_2, Cell player\_color) | Метод, предназначенный для проверки и установки шашки в новую позицию |
| void Board::draw(char\* buffer) | Метод, предназначенный для представления массива с шашками в виде шахматной доски. Выводится шахматная доска через asci символы и записывается в buffer |
| int Board::num\_chekers(Cell player\_color) | Метод, возвращающий количество оставшихся шашек у игрока с определённым цветом |

**Основные функции**

|  |  |
| --- | --- |
| **Название функции** | **Описание** |
| void \* white\_player(void \*) | Поток, в котором происходит обработка входящих данных от первого клиента. |
| void \* black\_player(void \*) | Поток, в котором происходит обработка входящих данных от второго клиента. |

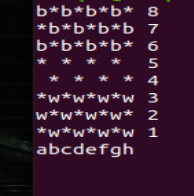
*2.3 Список системных вызовов операционной системы*

При написание кода использовался следующий набор функций

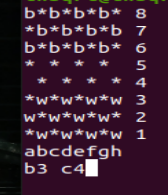
|  |  |
| --- | --- |
| **Название** | **Описание** |
| socket() | Служит для создания сокета в адресном пространстве ОС. Имеет три параметра. Первый параметр указывает к какому семейству протоколов относится создаваемый сокет, второй параметр определяет способ передачи данных по сети, а третий параметр определяет протокол, используемый для передачи данных. |
| connect() | Служит для установления соединения со стороны клиента. Имеет три параметра. Первый параметр указывает на сокет клиента, который будет использоваться для обмена данными с сервером, второй параметр содержит указатель на структуру с адресом сервера, а третий параметр содержит длину этой структуры. |
| recv() | Служит для чтения данных. Всего имеет четыре параметра. Первый параметр – дескриптор сокета, через который мы будет считывать данные. Второй параметр – буфер памяти, в который записываются данные. Третий параметр – размер данных в байтах, который мы хотим считать, а четвёртый параметр – набор битовых флагов, управляющих работой функции. |
| send() | Служит для отправки данных. Всего имеет четыре параметра. Первый параметр – дескриптор сокета, через который мы будет отправлять данные. Второй параметр – буфер памяти, который мы будем отправлять. Третий параметр – размер данных в байтах, который мы хотим отправить, а четвёртый параметр – набор битовых флагов, управляющих работой функции. |
| bind() | Явно связывает сокет с адресом в выбранном домене. Первый параметр – дескриптор сокета, который мы хотим привязать к заданному адресу. Второй параметр содержит указатель на структуру с адресом, третий параметр – длину этой структуры. |
| listen() | Прослушивает соединения на сокете. Позволяет показать готовность принимать соединения и задать лимит входящих соединений. Первым параметром указывается дескриптор сокета. Второй параметр определяет максимальную длину очереди входящих сообщений. |
| accept() | Функция создаёт для общения с клиентом новый сокет и возвращает его дескриптор. Первый параметр – слушающий сокет, на который приходят соединения. Второй параметр содержит структуру, в которую записывается адрес сокета клиента, который установил соединение с сервером, а в третий параметр записывается размер структуры. |
| pthread\_create() | Функция служит для создания нового потока управления. Всего имеет четыре параметра. Первый параметр ­– указатель на переменную, в которую будет помещен идентификатор созданного потока. Второй параметр – указатель на описатель атрибутов. Если указано NULL, то используются значения по умолчанию. Третий параметр – указатель на функцию, которая будет исполняться в рамках создаваемого потока управления. Четвёртый параметр – указатель на единственный аргумент функции, определённой третьим параметром. |
| pthread\_join() | Функция ожидания завершения потока управления. Имеет всего два параметра. Первый параметр – идентификатор потока управления, до завершения приостановлено выполнение данного потока. Второй параметр – указатель на переменную, в которую будет помещен статус окончания потока. |

*2.4 Пример работы приложения*

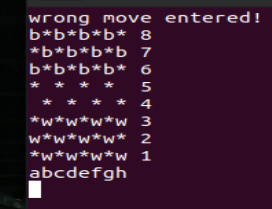
Вот что видит пользователь, когда игра начинается:



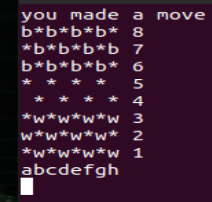
Ввод хода осуществляется через командную строку:



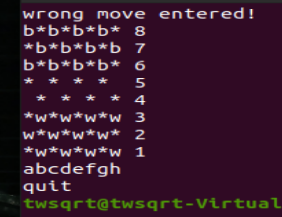
Если ход сделан неправильно, то сервер сообщает об ошибки и предлагает сделать ход ещё раз:



При верном ходе сервер подтверждает действие и обновляет экран:



Выход пользователя:



**РЕЗУЛЬТАТ РАБОТЫ**

В процессе выполнения этой курсовой я смог реализовать сетевую игру в Шашки, в которой вся логика сосредоточена на сервере, а клиент просто имеет методы для отправки и получения сообщений.

Курсовая сделана на системах семейства Unix/Linux. Код написан на языке c++ с использованием библиотеке <sys/socket.h>.

**ВЫВОД**

В результате выполнения курсовой работы я получил опыт разработки многопользовательских сетевых приложений под управлением операционной системы семейства Unix/Linux.

**ПРИЛОЖЕНИЕ А1**

*Файл server.cpp:*

#include <iostream>

#include <unistd.h>

#include <arpa/inet.h>

#include <sys/socket.h>

#include <netdb.h>

#include <netinet/in.h>

#include <string.h>

#include <pthread.h>

#define PORT 1601

const int BOARD\_SIZE = 8;

const int SCREEN\_BUFF\_SIZE = 128;

const int ROW\_CHEKERS\_NUM = 3;

const int SEND\_MESS\_SIZE = 32;

const char RECV\_QUIT\_MESSAGE[] = "quit";

const char INPUT\_ERROR\_MESS[SEND\_MESS\_SIZE] = "wrong move entered!";

const char CORRECT\_INPUT\_MESS[SEND\_MESS\_SIZE] = "you made a move";

const char WIN\_MESSAGE[SEND\_MESS\_SIZE] = "you win!";

const char LOSE\_MESSAGE[SEND\_MESS\_SIZE] = "you lose :(";

enum Cell

{

NONE,

CHEKER\_BLACK,

KING\_BLACK,

CHEKER\_WHITE,

KING\_WHITE,

BLACK = 0,

WHITE = 2,

CHEKER = 1,

KING

};

struct Cord

{

int x;

int y;

};

struct Message

{

char word\_1[5];

char word\_2[6];

};

class Board

{

private:

int num\_black;

int num\_white;

Cell map[BOARD\_SIZE\* BOARD\_SIZE];

public:

Cell & get(int x, int y);

Cell & get(Cord pos);

bool move(Cord pos\_1, Cord pos\_2);

void reset();

Board();

bool move\_by\_rule(Cord pos\_1, Cord pos\_2, Cell player\_color);

void draw(char\* buffer);

int num\_chekers(Cell player\_color);

};

Cell & Board::get(int x, int y)

{

return map[y\*BOARD\_SIZE + x];

}

Cell& Board::get(Cord pos)

{

return map[pos.y\*BOARD\_SIZE + pos.x];

}

bool Board::move(Cord pos\_1, Cord pos\_2)

{

if( this->get(pos\_2) == NONE)

{

std::swap(this->get(pos\_1), this->get(pos\_2));

return true;

}

return false;

}

void Board::reset()

{

for(int j, i =0; i < BOARD\_SIZE; i++)

{

for(j =0; j < ROW\_CHEKERS\_NUM; j++)

{

get(i, j) = (bool)((i +j)&1) ? CHEKER\_WHITE : NONE;

}

for(; j < BOARD\_SIZE - ROW\_CHEKERS\_NUM; j++)

{

get(i, j) = NONE;

}

for(; j<BOARD\_SIZE ; j++)

{

get(i, j) = NONE;//(bool)((i +j)&1) ? CHEKER\_BLACK : NONE;

}

}

get(5, 6) = CHEKER\_BLACK;

get(6,7) = CHEKER\_BLACK;

num\_white = num\_black = BOARD\_SIZE / 2 \*ROW\_CHEKERS\_NUM;

}

Board::Board()

{

reset();

}

bool Board::move\_by\_rule(Cord pos\_1, Cord pos\_2, Cell player\_color)

{

Cell enemy\_color = player\_color == WHITE ? BLACK : WHITE;

int direction = player\_color == WHITE ? 1 : -1;

if(get(pos\_2) != NONE || get(pos\_1) == get(pos\_2))

return false;

int dx= pos\_2.x - pos\_1.x;

int dy = pos\_2.y - pos\_1.y;

int sign\_dx = dx >0 ? 1 : -1;

int sing\_dy = dy >0 ? 1 : -1;

if(get(pos\_1) == CHEKER + player\_color)

{

bool flag = false;

if(dy == direction && abs(dx) == 1)

{

move(pos\_1, pos\_2);

flag = true;

}

if(dy == 2\*direction && abs(dx) == 2)

{

if(get(pos\_1.x +sign\_dx, pos\_1.y +1) == CHEKER + enemy\_color

|| get(pos\_1.x +sign\_dx, pos\_1.y +1) == KING + enemy\_color)

{

get(pos\_1.x + sign\_dx, pos\_1.y +1) = NONE;

move(pos\_1, pos\_2);

num\_black--;

flag = true;

}

}

if(flag && pos\_2.y == ( player\_color == WHITE ? BOARD\_SIZE -1 : 0 ) )

{

get(pos\_2) =(Cell)( KING + player\_color );

}

return flag;

}

if(get(pos\_1) == KING + player\_color)

{

if(abs(dx) != abs(dy))

return false;

if(abs(dx) == 2)

{

if(get(pos\_1.x +sign\_dx, pos\_1.y +sing\_dy) == CHEKER + enemy\_color

|| get(pos\_1.x +sign\_dx, pos\_1.y +sing\_dy) == KING + enemy\_color )

{

get(pos\_1.x + sign\_dx, pos\_1.y +sing\_dy) = NONE;

move(pos\_1, pos\_2);

num\_black--;

return true;

}

}

for(int i =0, count =1; i < abs(dx); i++)

{

if( get(pos\_2.x - sign\_dx\*i, pos\_2.y - sing\_dy\*i) != NONE)

return false;

}

move(pos\_1, pos\_2);

return true;

}

return false;

}

void Board::draw(char\* buffer)

{

static const char symbols[] =

{

' ', //NONE,

'b', //CHEKER\_BLACK,

'B', //KING\_BLACK,

'w', //CHEKER\_WHITE,

'W', //KING\_WHITE

'\*' //unused cells

};

for(int j, i = 0; i < BOARD\_SIZE; i++)

{

for(j = 0; j < BOARD\_SIZE; j++)

{

if((bool)((i + j)&1) )

buffer[i\*(BOARD\_SIZE+3) +j] = symbols[sizeof(symbols) -1];

else

buffer[i\*(BOARD\_SIZE+3) +j] = symbols[get(j, BOARD\_SIZE -i -1)];

}

buffer[i\*(BOARD\_SIZE+3) + j++] = ' ';

buffer[i\*(BOARD\_SIZE+3) + j++] = (char)(BOARD\_SIZE - i + '0');

buffer[i\*(BOARD\_SIZE+3) + j] = '\n';

}

for(int i =0; i < BOARD\_SIZE; i++)

{

buffer[BOARD\_SIZE\*(BOARD\_SIZE+3) +i] = (char)(i + 'a');

}

buffer[BOARD\_SIZE\*(BOARD\_SIZE+4)] = '\0';

}

int Board::num\_chekers(Cell player\_color)

{

return player\_color == WHITE ? num\_white : num\_black;

}

bool out\_of\_bounds(Cord pos)

{

return pos.x < 0 || pos.x > BOARD\_SIZE -1

|| pos.y < 0 || pos.y > BOARD\_SIZE -1;

}

int user\_1, user\_2;

bool player\_turn;

Board board;

void \* white\_player(void \*)

{

char screen\_buff[SCREEN\_BUFF\_SIZE];

Message message;

Cord select\_cell\_pos;

Cord new\_cell\_pos;

board.draw(screen\_buff);

while(board.num\_chekers(WHITE) > 0)

{

if( player\_turn)

{

send(user\_1, screen\_buff, SCREEN\_BUFF\_SIZE, 0);

recv(user\_1, &message, sizeof(message), 0);

if(strcmp(message.word\_1, RECV\_QUIT\_MESSAGE) == 0)

{

close(user\_1);

return NULL;

}

select\_cell\_pos.x = (int)(message.word\_1[0] - 'a');

select\_cell\_pos.y = (int)(message.word\_1[1] - '1');

new\_cell\_pos.x = (int)(message.word\_2[0] - 'a');

new\_cell\_pos.y = (int)(message.word\_2[1] - '1');

if(out\_of\_bounds(select\_cell\_pos)

|| out\_of\_bounds(new\_cell\_pos) )

{

send(user\_1, INPUT\_ERROR\_MESS, SEND\_MESS\_SIZE, 0);

}

else if( !board.move\_by\_rule(select\_cell\_pos, new\_cell\_pos, WHITE))

{

send(user\_1, INPUT\_ERROR\_MESS, SEND\_MESS\_SIZE, 0);

}

else

{

send(user\_1, CORRECT\_INPUT\_MESS, SEND\_MESS\_SIZE, 0);

}

player\_turn = !player\_turn;

}

}

send(user\_1, LOSE\_MESSAGE, SEND\_MESS\_SIZE, 0);

send(user\_2, WIN\_MESSAGE, SEND\_MESS\_SIZE, 0);

close(user\_1);

close(user\_2);

}

void \* black\_player(void \*)

{

char screen\_buff[SCREEN\_BUFF\_SIZE];

Message message;

Cord select\_cell\_pos;

Cord new\_cell\_pos;

board.draw(screen\_buff);

while(board.num\_chekers(BLACK) > 0)

{

if( ! player\_turn)

{

board.draw(screen\_buff);

send(user\_2, screen\_buff, SCREEN\_BUFF\_SIZE, 0);

recv(user\_2, &message, sizeof(message), 0);

if(strcmp(message.word\_1, RECV\_QUIT\_MESSAGE) == 0)

{

close(user\_2);

return NULL;

}

select\_cell\_pos.x = (int)(message.word\_1[0] - 'a');

select\_cell\_pos.y = (int)(message.word\_1[1] - '1');

new\_cell\_pos.x = (int)(message.word\_2[0] - 'a');

new\_cell\_pos.y = (int)(message.word\_2[1] - '1');

if(out\_of\_bounds(select\_cell\_pos)

|| out\_of\_bounds(new\_cell\_pos) )

{

send(user\_2, INPUT\_ERROR\_MESS, SEND\_MESS\_SIZE, 0);

}

else if( !board.move\_by\_rule(select\_cell\_pos, new\_cell\_pos, BLACK))

{

send(user\_2, INPUT\_ERROR\_MESS, SEND\_MESS\_SIZE, 0);

}

else

{

send(user\_2, CORRECT\_INPUT\_MESS, SEND\_MESS\_SIZE, 0);

}

player\_turn = !player\_turn;

}

}

send(user\_2, LOSE\_MESSAGE, SEND\_MESS\_SIZE, 0);

send(user\_1, WIN\_MESSAGE, SEND\_MESS\_SIZE, 0);

close(user\_2);

close(user\_1);

}

int main ()

{

int listener;

sockaddr\_in server\_address;

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

if(listener < 0)

{

std::cout << "SOCKET ERROR!" << std::endl;

return 0;

}

std::cout << "SOCKET WAS CREATED" << std::endl;

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

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

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

socklen\_t address\_size = sizeof(server\_address);

int ret = bind (listener, (sockaddr \*)&server\_address, address\_size);

if(ret < 0)

{

std::cout << "BIND ERROR!" << std::endl;

return 0;

}

listen(listener, 2);

user\_1 = accept(listener, NULL , NULL);

user\_2 = accept(listener, NULL, NULL);

if(user\_1 < 0 || user\_2 <0)

{

close(user\_1);

close(user\_2);

std::cout << "USER 1 ACCEPT ERROR!" << std::endl;

return 0;

}

pthread\_t usr\_1\_thread, usr\_2\_thread;

player\_turn = true;

int res1 = pthread\_create(&usr\_1\_thread, NULL, white\_player, NULL);

int res2 = pthread\_create(&usr\_2\_thread, NULL, black\_player, NULL);

if(res1 != 0 || res2 != 0)

{

std::cout << "THREAD ERROR!" <<std::endl;

close(user\_1);

close(user\_2);

return 0;

}

pthread\_join(usr\_1\_thread, NULL);

pthread\_join(usr\_2\_thread, NULL);

}

**ПРИЛОЖЕНИЕ А2**

Файл user.cpp:

#include <iostream>

#include <unistd.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <string.h>

#define PORT 1601

const int RECV\_BUFF = 32;

const int SCREEN\_BUFF\_SIZE = 128;

const char QUIT\_MESSAGE[] = "quit";

struct Message

{

char word\_1[5];

char word\_2[6];

};

int main()

{

int sock;

sockaddr\_in server\_address;

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

if (sock < 0)

{

std::cout << "SOCKET ERROR!" << std::endl;

return 0;

}

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

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

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

socklen\_t address\_size = sizeof(server\_address);

int ret = connect (sock, (sockaddr \*)&server\_address, address\_size);

if( ret < 0)

{

std::cout << "CONNECT ERROR!" << std::endl;

return 0;

}

Message message;

char recv\_buff[RECV\_BUFF];

char screen\_buff[SCREEN\_BUFF\_SIZE];

bool connected = true;

while(connected)

{

recv(sock, screen\_buff, SCREEN\_BUFF\_SIZE, 0);

std::cout << screen\_buff << std::endl;

std::cin >> message.word\_1;

if(strcmp(message.word\_1, QUIT\_MESSAGE) == 0)

{

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

close(sock);

connected = false;

}

else

{

std::cin >> message.word\_2;

std::ios\_base::sync\_with\_stdio(0);

std::cin.ignore(std::cin.rdbuf() -> in\_avail());

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

recv(sock, recv\_buff, RECV\_BUFF, 0);

system("clear");

std::cout << recv\_buff <<std::endl;

}

}

}