ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ

**РАСЧЕТНО-ГРАФИЧЕСКОЕ ЗАДАНИЕ**

по дисциплине “Распределённые системы и технологии”

на тему

**Проектирование и разработка гибридного (параллельного и распределённого) приложения «Игра в крестики-нолики»**

|  |  |
| --- | --- |
| Выполнил студент | Марков В.А. |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | МГ-165 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | д.т.н. профессор С.Н. Мамойленко |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2016

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ](#__RefHeading___Toc337081083) 3

[1. Асинхронные сокеты](#__RefHeading___Toc337081084)……………………………………………………………...4

2. Отказоустойчивость……………………………………………………………….4

3. Постановка задачи…………………………………………………………………5

4. Реализация…………………………………………………………………………6

[ЗАКЛЮЧЕНИЕ 11](#__RefHeading___Toc337081086)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 12](#__RefHeading___Toc337081087)

[ПРИЛОЖЕНИЕ 13](#__RefHeading___Toc337081088)

ВВЕДЕНИЕ

Большинство задач, которые приходится решать программисту в повседневной жизни, связано с выполнением локальных действий в системе.

Обычно они не выходят за рамки управления мышью, клавиатурой, экраном и файловой системой. Гораздо труднее заставить взаимодействовать несколько программ, которые работают на разных компьютерах, подключенных к сети.

Сетевое программирование, несомненно, полезно для опыта, так как приходится координировать работу множества компонентов приложения и тщательно распределять обязанности между ними. Фундаментальной единицей всего сетевого программирования в Linux (и большинстве других операционных систем) является сокет. Так же, как функции файлового ввода вывода определяют интерфейс взаимодействия с файловой системой, сокет соединяет программу с сетью. С его помощью она посылает и принимает сообщения.

Создавать сетевые приложения труднее, чем даже заниматься многозадачным программированием, так как круг возникающих проблем здесь гораздо шире. Необходимо обеспечивать максимальную производительность, координировать обмен данными и управлять вводом выводом. В рамках данного курсового проекта реализовано клиент-серверное приложение – игра «Крестики-нолики».

1. Асинхронные сокеты Linux epoll()

Epoll (extended poll) – ЫAPI мультиплексированного ввода-вывода, предоставляемого Linux для приложений. API позволяет приложения осуществлять мониторинг нескольких открытых файловых дескрипторов (которые могут быть файлами, устройствами или сокетами, в том числе сетевыми), для того, чтобы узнать, готово ли устройство для продолжения ввода (вывода).

Epoll планировался как более эффективная замена вызовам select() и poll(), определёнными в POSIX. Epoll может предоставить более эффективный механизм для приложений, обрабатывающих большое количество одновременно открытых соединений – со сложностью O(1) в отличие от стандартного механизма, обладающего сложностью O(n). Epoll аналогичен системе Kqueue из FreeBSD и также представляет собой объект ядра, предоставляемый для работы в пространстве пользователя в виде файлового дескриптора [1].

2. Отказоустойчивость

Отказоустойчивая архитектура с точки зрения инженерии – это метод проектирования отказоустойчивых систем, которые способны продолжать выполнение запланированных операций (возможно, с понижением эффективности) при отказе их компонентов. Термин часто используется для описания компьютерных систем, спроектированных продолжать работу в той или иной степени, с возможным уменьшением пропускной способности или увеличением времени отклика, в случае отказа части системы. Это означает, что система в целом не прекратит свою работу при возникновении проблем с аппаратной или программной частью. Если каждый компонент системы может продолжать работать при отказе одной из его составляющих, то вся система, в свою очередь, также продолжает работать [2].

Впервые идея включения избыточных частей для увеличения надежности системы была высказана Джоном фон Нейманом в 1950-х годах.

Существует два типа избыточности: пространственная и временная:

* Избыточность пространства реализуется путем введения дополнительных компонентов, функций или данных, которые не нужны при безотказном функционировании. Дополнительные (избыточные) компоненты могут быть аппаратными, программными и информационными.
* Временная избыточность реализуется путем повторных вычислений или отправки данных, после чего результат сравнивается с сохранённой копией предыдущего.

3. Постановка задачи

В ходе выполнения расчетно-графического задания необходимо спроектировать и разработать гибридное (параллельное и распределённое) программное обеспечение, реализующие сетевую многопользовательскую игру «крестики-нолики». Правила игры классические (поле 3 на 3, в каждой игре участвуют два игрока).

Разрабатываемое программное обеспечение должно иметь возможность реализовать игровую ситуацию как на одной ЭВМ, так и с использованием распределённой среды.

В случае реализации игры на одной ЭВМ одним игроком является пользователь приложения (интерактивный режим), второй игрок эмулируется (как параллельно выполняющийся поток).

При реализации игры в распределённом режиме управлением игрой должен заниматься выделенный сервер (отдельный процесс). На стороне клиента должно быть обеспечено одновременное прохождение и игры и сетевой чат. Операционная система, под управлением которой будет функционировать разрабатываемое приложение, выбирается студентом (допускается реализация кроссплатформенного приложения). В отчете по выполнению расчетно-графического задания должно присутствовать описание всех этапов проектирования программного обеспечения (описание общей архитектуры приложения, разработка параллельного алгоритма эмулятора игрока на стороне клиента и т.п.) и описание всех выбранных для его реализации технологий и программных средств. Также должны быть представлены скриншоты, демонстрирующие работу созданного программного обеспечения.

4. Реализация

**/\***

**\* Перечисление типов игровых сообщений**

**\* AUTH - авторизация**

**\* GAME - игра**

**\* CHAT - чат**

**\* RESR – резервный сервер**

**\* EVN - события**

**\*/**

**enum MSG { AUTH, GAME, CHAT, RESR, EVN };**

**/\***

**\* Перечисление действий**

**\* YES - да**

**\* NO – нет**

**\*/**

**enum ACT { YES, NO };**

**/\***

**\* Перечисление меток**

**\* ZERO - игрок-ноль**

**\* CROSS – игрок-крест**

**\* EMPTY – пустая игровая комната**

**\* FULL - полная игровая комната**

**\*/**

**enum MRK { ZERO, CROSS, EMPTY, FULL };**

**/\***

**\* Перечисление событий**

**\* TECH\_WIN – техническая победа**

**\* UPD - обновление**

**\*/**

**enum EVN { TECH\_WIN, UPD };**

**typedef struct {**

**int room\_state;**

**int room\_number;**

**int room\_players[2];**

**int room\_field[3][3];**

**} room\_info; // описание игровой комнаты**

**typedef struct {**

**int request;**

**room\_info game\_rooms[4];**

**} resr\_msg; // резервное сообщение**

**typedef struct {**

**int new\_game;**

**int number;**

**int player;**

**int start**

**} auth\_msg; // сообщение авторизации**

**typedef struct {**

**int number;**

**int player;**

**int row;**

**int col;**

**} game\_msg; // сообщение игровой сессии**

**typedef struct {**

**int number;**

**int player;**

**char string[256];**

**} chat\_msg; // сообщение чата**

**typedef struct {**

**int event;**

**} evnt\_msg; // сообщение события**

**typedef struct {**

**int type union**

**{**

**evnt\_msg evnt;**

**auth\_msg auth;**

**game\_msg game;**

**chat\_msg chat;**

**resr\_msg resr;**

**};**

**} common\_msg; // основное сообщение**

Листинг 1 – Реализация протокола общения между сервером и клиентом

5. Игровые ситуации

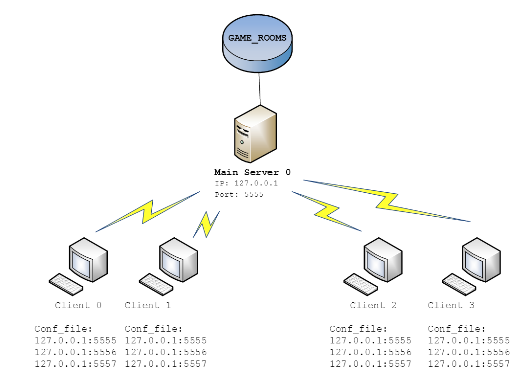


Рисунок 1 – Стандартная игровая ситуация

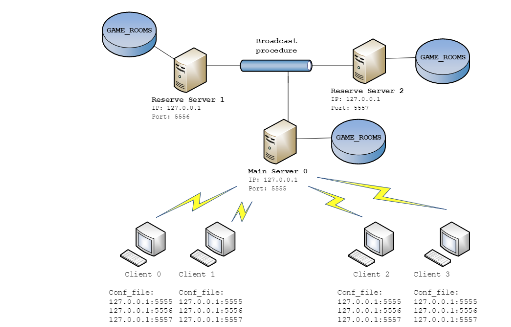


Рисунок 2 – Игровая ситуация с подключенными резервными серверами

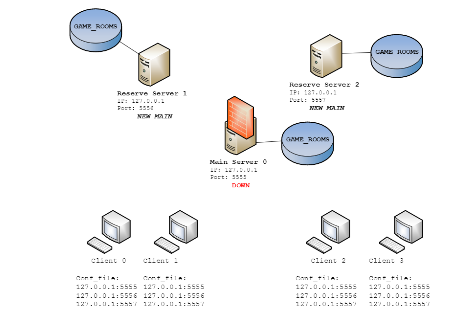


Рисунок 3 – Игровая ситуация с основным сервером потеряно соединение

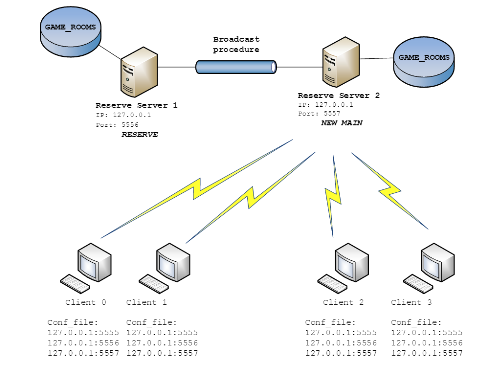


Рисунок 4 – Игровая ситуация резервный сервер берет на себя роль основного

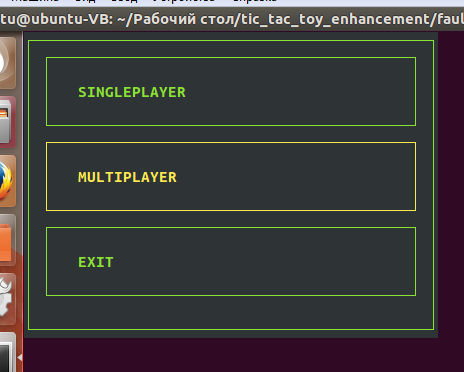


Рисунок 4 – Главное меню

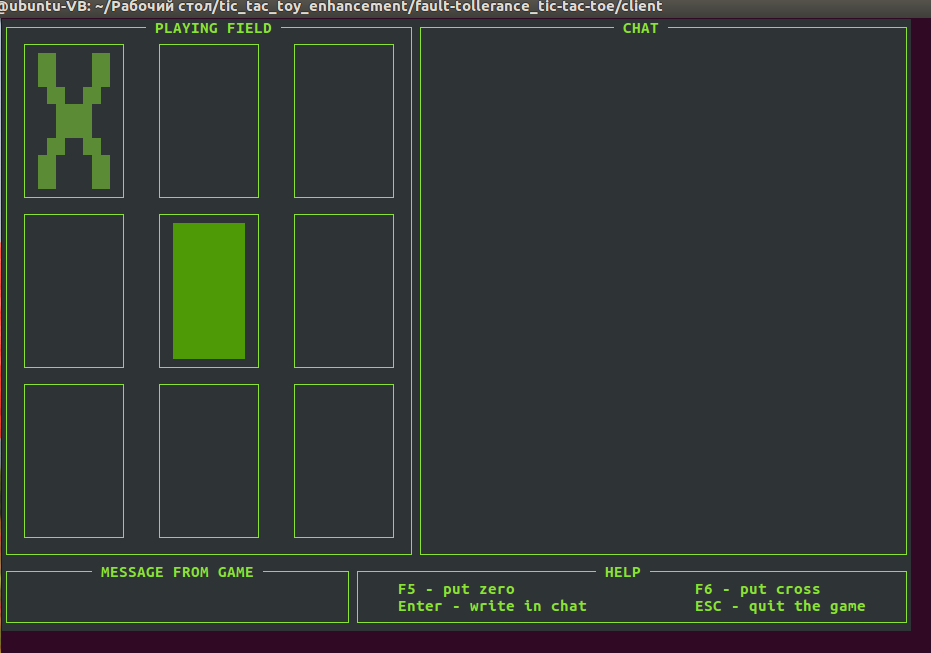


Рисунок 5 – Внешний вид разработанного приложения

ЗАКЛЮЧЕНИЕ

В результате выполнения расчетно-графического задания спроектировано и разработано гибридное (параллельное и распределённое) программное обеспечение, реализующие сетевую многопользовательскую игру «крестики-нолики».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ru.wikipedia.org/wiki/Epoll
2. ru.wikipedia.org/wiki/Отказоустойчивость
3. www.opennet.ru/man
4. Уолтон Ш. Создание сетевых приложений в среде Linux.: Пер. с англ.— М.: Вильямс, 2001.
5. Иванов Н.Н. Программирование в Linux. Самоучитель. –СПб.: БХВ-Петербург, 2007. – 416 с.

ПРИЛОЖЕНИЕ

**Client.c**

#include "header.h"

int CONF\_LIST[] = { 2345, // Main server port

2346, // First reserve server port

2347 }; // Second reserve server port

int GAME\_ROOM; // gome room number

int PLAYER; // player X or O

int SERVER\_FD; // server

// print network message

void print\_msg(common\_msg msg)

{

switch(msg.type)

{

case AUTH:

printf("AUTH:\n");

printf("room number %d\n", msg.auth.number);

if (msg.auth.player == ZERO) {

printf("player: ZERO\n");

} else {

printf("player: CROSS\n");

}

if (msg.auth.start == NO ) {

printf("state: wait\n");

} else {

printf("state: play\n");

}

break;

}

}

// initialize socket

static struct sockaddr\_in configure\_socket(struct sockaddr\_in server, int port)

{

bzero(&server, sizeof(server));

server.sin\_family = AF\_INET;

// переводим в сетевой порядок

server.sin\_port = htons(port);

// преобразует строку с адресом в двоичную форму с сетевым порядком

inet\_aton(SERVER\_IP, &(server.sin\_addr));

return server;

}

// try to connect to server

int try\_to\_connect(int server\_fd, int mode)

{

int i, fail\_conn;

struct sockaddr\_in server\_addr;

common\_msg msg;

server\_fd = socket(AF\_INET, SOCK\_STREAM, 0); // initialize socket

if (server\_fd < 0 ) {

fprintf(stderr, "error: can't create socket\n");

return -1;

}

// Loop for the try to connect (main or reserve0 or reserve1)

for (i = 0, fail\_conn = 0; i < NUM\_SERVER; i++) {

// Configure socket by differnt port

server\_addr = configure\_socket(server\_addr, CONF\_LIST[i]);

// Try to connect

if (connect(server\_fd, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) {

// Fail, reconnect to another

//fprintf(stderr, "error: can't connect to %s:%d. try to another...\n", SERVER\_IP, CONF\_LIST[i]);

fail\_conn++;

} else {

// Success

switch(mode)

{

case 0: // branch new connection

printf("connected to %s:%d\n",SERVER\_IP, CONF\_LIST[i]);

msg.type = AUTH;

msg.auth.new\_game = YES;

write(server\_fd, &msg, sizeof(msg));

bzero(&msg, sizeof(msg));

read(server\_fd, &msg, sizeof(msg));

// set field game room and player X or O

GAME\_ROOM = msg.auth.number;

PLAYER = msg.auth.player;

print\_msg(msg);

break;

case 1: // branch restore connection

displayMsgChat("connection restore...\n");

msg.type = AUTH;

msg.auth.new\_game = NO;

msg.auth.number = GAME\_ROOM;

msg.auth.player = PLAYER;

write(server\_fd, &msg, sizeof(msg));

break;

default:

break;

}

break;

}

if (fail\_conn == NUM\_SERVER) {

// All server are down

fprintf(stderr, "error: all servers are down, shutting down...\n");

close(server\_fd);

return -1;

}

}

return server\_fd;

}

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

{

switch (menu\_session())

{

case 0:

single\_game\_session();

break;

case 1:

{

SERVER\_FD = try\_to\_connect(SERVER\_FD, 0);

if (SERVER\_FD == -1) {

return -1;

}

game\_session(SERVER\_FD, 1);

close(SERVER\_FD);

}

break;

case 2:

break;

}

return 0;

}

**Single\_player.c**

#include "header.h"

// библиотеки для рисования интерфейса считывания клавиш

#include "../lib/graphics/myTerm.h"

#include "../lib/graphics/myBigChars.h"

#include "../lib/graphics/myReadKey.h"

#include <errno.h>

#include <netdb.h>

#include <stdio.h>

#include <fcntl.h>

#include <string.h>

#include <stdlib.h>

#include <unistd.h>

#include <strings.h>

#include <sys/epoll.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <pthread.h>

#include <signal.h> // для обработки сигнала прерывания

#include <sys/poll.h>

#include "../lib/message.h"

int gameField[3][3] = { { EMPTY, EMPTY, EMPTY },

{ EMPTY, EMPTY, EMPTY },

{ EMPTY, EMPTY, EMPTY } };

int is\_finished\_game = 0;

int previous\_player\_move = EMPTY;

pthread\_mutex\_t game\_mutex;

void lock\_gameField()

{

pthread\_mutex\_lock(&game\_mutex);

}

void unlock\_gameField()

{

pthread\_mutex\_unlock(&game\_mutex);

}

void\* ctlAiInteraction(void \*arg)

{

int chPlayer = CROSS;

while(1) {

sleep(1);

try\_to\_make\_move\_ai(chPlayer);

}

return 0;

}

int gamePlay(int gameState[3][3])

{

/\* case7 case8

\ /

\ c4 c5 c6 /

\ || || || /

\ \/ \/ \/ /

case 1 -----> | x | o | x |

case 2 -----> | o | x | o |

case 3 -----> | o | x | x |

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* проверка 8 случаев для нуля \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* case 1 - 3 \*/

if (gameState[0][0] == ZERO && gameState[0][1] == ZERO && gameState[0][2] == ZERO) {

return ZERO;

}

if (gameState[1][0] == ZERO && gameState[1][1] == ZERO && gameState[1][2] == ZERO) {

return ZERO;

}

if (gameState[2][0] == ZERO && gameState[2][1] == ZERO && gameState[2][2] == ZERO) {

return ZERO;

}

/\* case 4 - 6 \*/

if (gameState[0][0] == ZERO && gameState[1][0] == ZERO && gameState[2][0] == ZERO) {

return ZERO;

}

if (gameState[0][1] == ZERO && gameState[1][1] == ZERO && gameState[2][1] == ZERO) {

return ZERO;

}

if (gameState[0][2] == ZERO && gameState[1][2] == ZERO && gameState[2][2] == ZERO) {

return ZERO;

}

/\* case 7 - 8 \*/

if (gameState[0][0] == ZERO && gameState[1][1] == ZERO && gameState[2][2] == ZERO) {

return ZERO;

}

if (gameState[0][2] == ZERO && gameState[1][1] == ZERO && gameState[2][0] == ZERO) {

return ZERO;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* проверка 8 случаев для крестика \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\* case 1 - 3 \*/

if (gameState[0][0] == CROSS && gameState[0][1] == CROSS && gameState[0][2] == CROSS) {

return CROSS;

}

if (gameState[1][0] == CROSS && gameState[1][1] == CROSS && gameState[1][2] == CROSS) {

return CROSS;

}

if (gameState[2][0] == CROSS && gameState[2][1] == CROSS && gameState[2][2] == CROSS) {

return CROSS;

}

/\* case 4 - 6 \*/

if (gameState[0][0] == CROSS && gameState[1][0] == CROSS && gameState[2][0] == CROSS) {

return CROSS;

}

if (gameState[0][1] == CROSS && gameState[1][1] == CROSS && gameState[2][1] == CROSS) {

return CROSS;

}

if (gameState[0][2] == CROSS && gameState[1][2] == CROSS && gameState[2][2] == CROSS) {

return CROSS;

}

/\* case 7 - 8 \*/

if (gameState[0][0] == CROSS && gameState[1][1] == CROSS && gameState[2][2] == CROSS) {

return CROSS;

}

if (gameState[0][2] == CROSS && gameState[1][1] == CROSS && gameState[2][0] == CROSS) {

return CROSS;

}

// проверка на ничью

if (gameState[0][0] != EMPTY && gameState[0][1] != EMPTY && gameState[0][2] != EMPTY &&

gameState[1][0] != EMPTY && gameState[1][1] != EMPTY && gameState[1][2] != EMPTY &&

gameState[2][0] != EMPTY && gameState[2][1] != EMPTY && gameState[2][2] != EMPTY) {

return DRAW;

}

return EMPTY;

}

int winGame(int winner)

{

if (winner == EMPTY) {

return 0;

}

mt\_setstdcolor();

mt\_clrscr();

mt\_setbgcolor(clBlack);

mt\_setfgcolor(clGreen);

mt\_gotoXY(15, 17);

switch(winner)

{

case ZERO:

write(1, "WON ZEROS", 9);

return 1;

case CROSS:

write(1, "WON CROSSES", 11);

return 1;

case DRAW:

write(1, "PLAYED IN DRAW!", 15);

return 1;

default:

return 0;

}

return 0;

}

// отрисовывет значение клеток и выделяет нужную

void drawGameField(int ROW, int COL)

{

int big[2];

/\* позиция символа по Х \*/

unsigned int posX;

unsigned int i;

/\* 1 - нолик, 2 - крестик \*/

for (i = 0, posX = 5; i < 3; i++, posX = posX + 15) {

/\* выделим нужную ячейку \*/

if (ROW == 0 && COL == i) {

if (gameField[0][i] == ZERO) {

bc\_setbigcharpos(big, 3, posX, 'o', clBlack, clGreen);

}

if (gameField[0][i] == CROSS) {

bc\_setbigcharpos(big, 3, posX, 'x', clBlack, clGreen);

}

if (gameField[0][i] == EMPTY) {

bc\_setbigcharpos(big, 3, posX, '\*', clBlack, clGreen);

}

} else {

if (gameField[0][i] == ZERO) {

bc\_setbigcharpos(big, 3, posX, 'o', clGreen, clBlack);

}

if (gameField[0][i] == CROSS) {

bc\_setbigcharpos(big, 3, posX, 'x', clGreen, clBlack);

}

/\*\*\*\*\*/

if (gameField[0][i] == EMPTY) {

bc\_setbigcharpos(big, 3, posX, '\*', clGreen, clBlack);

}

}

}

for (i = 0, posX = 5; i < 3; i++, posX = posX + 15) {

/\* выделим нужную ячейку \*/

if (ROW == 1 && COL == i) {

if (gameField[1][i] == ZERO) {

bc\_setbigcharpos(big, 13, posX, 'o', clBlack, clGreen);

}

if (gameField[1][i] == CROSS) {

bc\_setbigcharpos(big, 13, posX, 'x', clBlack, clGreen);

}

if (gameField[1][i] == EMPTY) {

bc\_setbigcharpos(big, 13, posX, '\*', clBlack, clGreen);

}

} else {

if (gameField[1][i] == ZERO) {

bc\_setbigcharpos(big, 13, posX, 'o', clGreen, clBlack);

}

if (gameField[1][i] == CROSS) {

bc\_setbigcharpos(big, 13, posX, 'x', clGreen, clBlack);

}

/\*\*\*\*\*/

if (gameField[1][i] == EMPTY) {

bc\_setbigcharpos(big, 13, posX, '\*', clGreen, clBlack);

}

}

}

for (i = 0, posX = 5; i < 3; i++, posX = posX + 15) {

/\* выделим нужную ячейку \*/

if (ROW == 2 && COL == i) {

if (gameField[2][i] == ZERO) {

bc\_setbigcharpos(big, 23, posX, 'o', clBlack, clGreen);

}

if (gameField[2][i] == CROSS) {

bc\_setbigcharpos(big, 23, posX, 'x', clBlack, clGreen);

}

if (gameField[2][i] == EMPTY) {

bc\_setbigcharpos(big, 23, posX, '\*', clBlack, clGreen);

}

} else {

if (gameField[2][i] == ZERO) {

bc\_setbigcharpos(big, 23, posX, 'o', clGreen, clBlack);

}

if (gameField[2][i] == CROSS) {

bc\_setbigcharpos(big, 23, posX, 'x', clGreen, clBlack);

}

/\*\*\*\*\*/

if (gameField[2][i] == EMPTY) {

bc\_setbigcharpos(big, 23, posX, '\*', clGreen, clBlack);

}

}

}

}

int try\_to\_make\_move(int rowField, int colField, int player)

{

lock\_gameField();

if (gameField[rowField][colField] == EMPTY && previous\_player\_move != player) {

previous\_player\_move = player;

gameField[rowField][colField] = player; // ставим в массиве нолик

drawGameField(rowField, colField); // отрисовываем интерфейс заново

// проверяем на выигрышную ситуацию (другой клиент проерит в своей потоке)

is\_finished\_game = winGame(gamePlay(gameField));

} else if (gameField[rowField][colField] != EMPTY){

mi\_writeMessage("This place is taken!");

} else if (previous\_player\_move == player) {

mi\_writeMessage("CROSS move!");

}

unlock\_gameField();

return 0;

}

void try\_to\_make\_move\_ai(int player)

{

lock\_gameField();

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (gameField[i][j] == EMPTY && previous\_player\_move != player) {

previous\_player\_move = player;

gameField[i][j] = player; // ставим в массиве нолик

drawGameField(i, j); // отрисовываем интерфейс заново

unlock\_gameField();

return;

}

}

}

is\_finished\_game = winGame(gamePlay(gameField));

unlock\_gameField();

}

// main single player game routine

int single\_game\_session()

{

int key;

int chPlayer = ZERO;

pthread\_t ai\_player;

int rowField = 0;

int colField = 0;

char msgChat[250] = "";

int len;

// Draw main part of interface

mi\_drawMainInterface();

drawGameField(0, 0);

// Save terminal settings, change mode, hide cursor

rk\_mytermsave();

rk\_mytermregime(0, 0, 1, 0, 1);

mt\_setcursor(0);

// Setup color palete

mt\_setbgcolor(clBlack);

mt\_setfgcolor(clGreen);

int unused;

pthread\_create(&ai\_player, 0, ctlAiInteraction, (void\*)&unused);

int whoWin;

char msgSign[250];

if (chPlayer == 1) {

mi\_writeMessage("You are playing ZEROS");

} else if (chPlayer == 2) {

mi\_writeMessage("You are playing CROSSES");

}

while (is\_finished\_game == 0 && rk\_readkey(&key) == 0 && key != K\_ESC) {

switch (key)

{

case K\_R:

// not supproted

break;

case K\_T:

// not supproted

break;

case K\_UP:

if (rowField != 0) {

rowField--;

drawGameField(rowField, colField);

}

break;

case K\_DOWN:

if (rowField < 2) {

rowField++;

drawGameField(rowField, colField);

}

break;

case K\_LEFT:

if (colField != 0) {

colField--;

drawGameField(rowField, colField);

}

break;

case K\_RIGHT:

if (colField < 2) {

colField++;

drawGameField(rowField, colField);

}

break;

case K\_ENTER:

break;

case K\_F5: // place ZERO

try\_to\_make\_move(rowField, colField, chPlayer);

break;

case K\_F6: // place CROSS

try\_to\_make\_move(rowField, colField, chPlayer);

break;

}

}

pthread\_cancel(ai\_player);

rk\_mytermrestore();

mt\_setcursor(1);

mt\_setstdcolor();

mt\_gotoXY(39, 1);

return 0;

}

void draw\_main\_menu()

{

int tty; /\* хранит номер дескриптора \*/

tty = open("/dev/tty", O\_RDWR); /\* открытие файла терминала в режиме чтение/запись \*/

if (tty == -1) { /\* проверка на открытие \*/

fprintf(stderr, "\nmi\_drawMainInterface()\n: Can not open tty\n");

close(tty);

return;

}

mt\_clrscr();

mt\_setbgcolor(clBlack);

mt\_setfgcolor(clGreen);

bc\_box(1, 1, 18, 46);

bc\_box(2, 3, 5, 42);

mt\_setfgcolor(clYellow);

bc\_box(7, 3, 5, 42);

mt\_setfgcolor(clGreen);

bc\_box(12, 3, 5, 42);

mt\_gotoXY(4, 6);

write(tty, " SINGLEPLAYER ", 14);

mt\_setfgcolor(clYellow);

mt\_gotoXY(9, 6);

write(tty, " MULTIPLAYER ", 13);

mt\_setfgcolor(clGreen);

mt\_gotoXY(14, 6);

write(tty, " EXIT ", 6);

return;

}

void draw\_main\_menu\_field(int row)

{

int tty; /\* хранит номер дескриптора \*/

tty = open("/dev/tty", O\_RDWR); /\* открытие файла терминала в режиме чтение/запись \*/

if (tty == -1) { /\* проверка на открытие \*/

fprintf(stderr, "\nmi\_drawMainInterface()\n: Can not open tty\n");

close(tty);

return;

}

if (row == 0) {

mt\_setfgcolor(clYellow);

bc\_box(2, 3, 5, 42);

mt\_gotoXY(4, 6);

write(tty, " SINGLEPLAYER ", 14);

mt\_setfgcolor(clGreen);

bc\_box(7, 3, 5, 42);

bc\_box(12, 3, 5, 42);

mt\_gotoXY(9, 6);

write(tty, " MULTIPLAYER ", 13);

mt\_gotoXY(14, 6);

write(tty, " EXIT ", 6);

} else if (row == 1) {

mt\_setfgcolor(clYellow);

bc\_box(7, 3, 5, 42);

mt\_gotoXY(9, 6);

write(tty, " MULTIPLAYER ", 13);

mt\_setfgcolor(clGreen);

bc\_box(2, 3, 5, 42);

bc\_box(12, 3, 5, 42);

mt\_gotoXY(4, 6);

write(tty, " SINGLEPLAYER ", 14);

mt\_gotoXY(14, 6);

write(tty, " EXIT ", 6);

} else {

mt\_setfgcolor(clYellow);

bc\_box(12, 3, 5, 42);

mt\_gotoXY(14, 6);

write(tty, " EXIT ", 6);

mt\_setfgcolor(clGreen);

bc\_box(2, 3, 5, 42);

bc\_box(7, 3, 5, 42);

mt\_gotoXY(4, 6);

write(tty, " SINGLEPLAYER ", 14);

mt\_gotoXY(9, 6);

write(tty, " MULTIPLAYER ", 13);

}

}

int menu\_session()

{

int key;

int run\_flag = -1;

int rowField = 1;

// Draw main part of interface

draw\_main\_menu();

// Save terminal settings, change mode, hide cursor

rk\_mytermsave();

rk\_mytermregime(0, 0, 1, 0, 1);

mt\_setcursor(0);

// Setup color palete

mt\_setbgcolor(clBlack);

mt\_setfgcolor(clGreen);

while (rk\_readkey(&key) == 0 && key != K\_ESC) {

switch (key)

{

case K\_UP:

if (rowField != 0) {

rowField--;

draw\_main\_menu\_field(rowField);

}

break;

case K\_DOWN:

if (rowField != 2) {

rowField++;

draw\_main\_menu\_field(rowField);

}

break;

case K\_ENTER:

if (rowField == 0) {

run\_flag = 0;

} else if (rowField == 1) {

run\_flag = 1;

} else {

run\_flag = 2;

}

break;

}

if (key == K\_ENTER) {

break;

}

}

rk\_mytermrestore();

mt\_setcursor(1);

mt\_setstdcolor();

mt\_gotoXY(39, 1);

mt\_clrscr();

return run\_flag;

}

**Server.c**

#include "header.h"

// Global variables

extern int SERVER\_STATE;

room\_info GAME\_ROOMS[NUM\_GAMES];

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

{

char status[256];

int epoll\_fd;

int server\_fd;

int server\_port = 5555;

struct sockaddr\_in server\_addr;

struct sockaddr\_in client\_addr;

struct epoll\_event \*events;

socklen\_t client\_len = sizeof(client\_addr);

if (argc < 3) {

err\_msg("", "not enought argumens");

printf("usage: ./server [port] [state]\n");

return 1;

}

// Initialize server state

SERVER\_STATE = atoi(argv[2]);

// Inintialize server port

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

// Initialize game rooms

initialize\_game\_rooms(GAME\_ROOMS, NUM\_GAMES);

print\_game\_rooms(GAME\_ROOMS, NUM\_GAMES);

if (SERVER\_STATE == 0) {

log\_msg("[LOG] CREATE MAIN SERVER - OK", 1);

create\_main\_server(server\_port, argv[1]);

} else if (SERVER\_STATE == 1) {

log\_msg("[LOG] CREATE RESERVE SERVER - OK", 1);

create\_reserve\_server(server\_port, argv[1]);

} else {

err\_msg(" Failed to associate server state.", "");

return -1;

}

return 0;

}