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

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

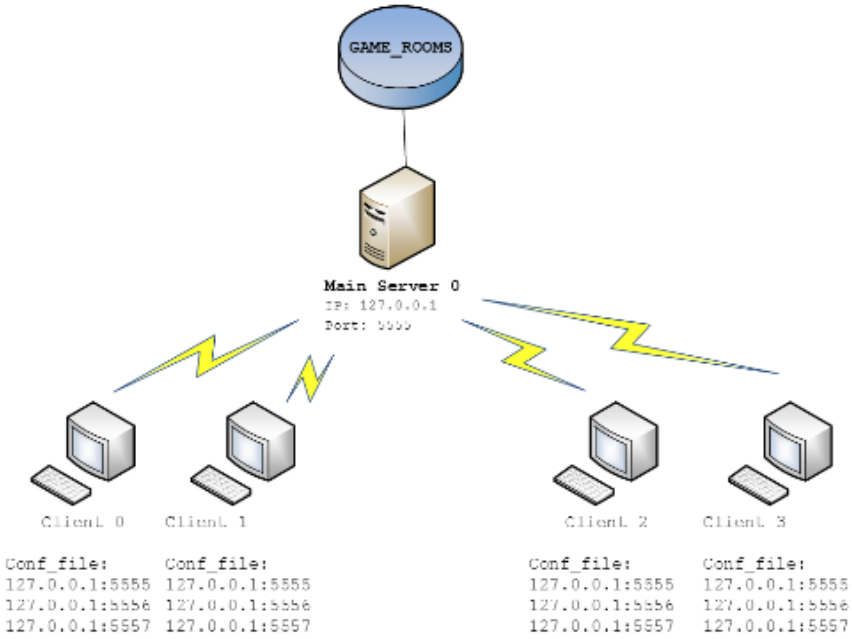

Рисунок 1 – Стандартная игровая ситуация

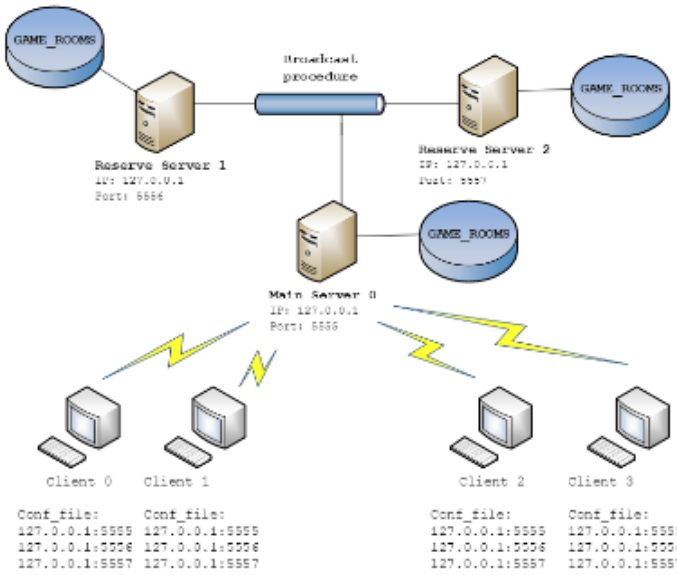

Рисунок 2 – Игровая ситуация с подключенными резервными серверами

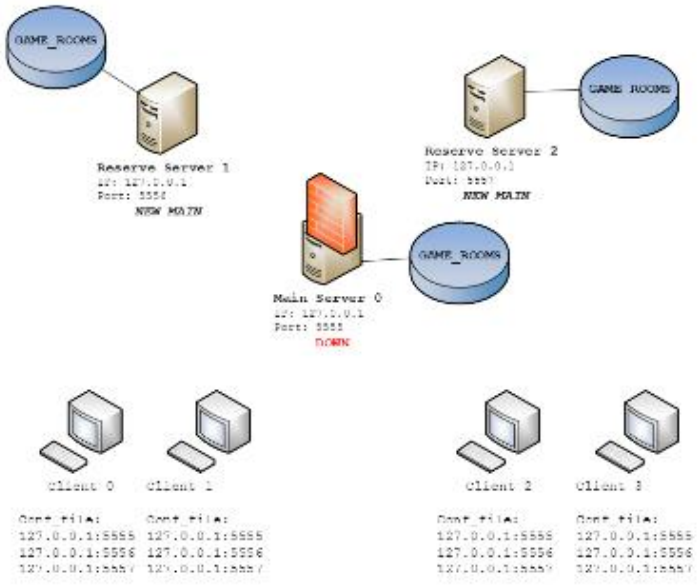

Рисунок 3 – Игровая ситуация с основным сервером потеряно соединение

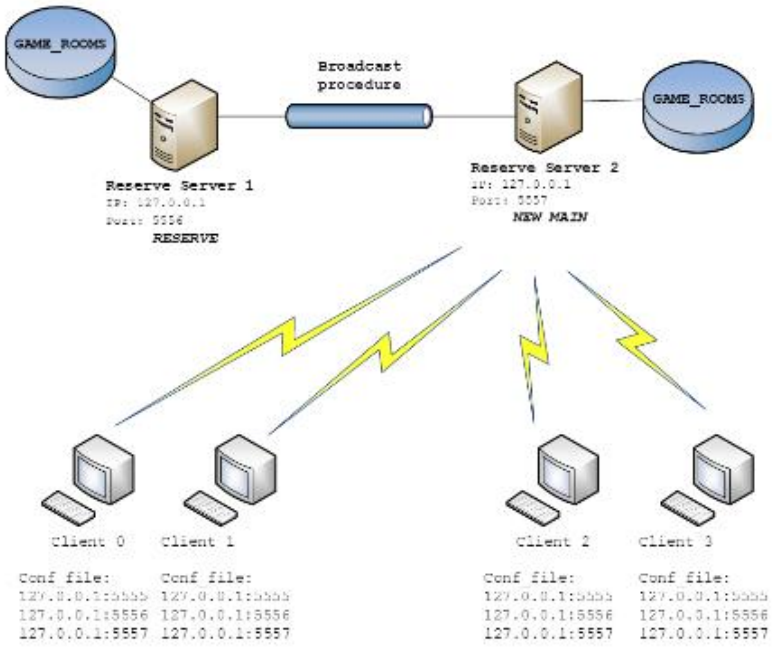

Рисунок 4 – Игровая ситуация резервный сервер берет на себя роль основного

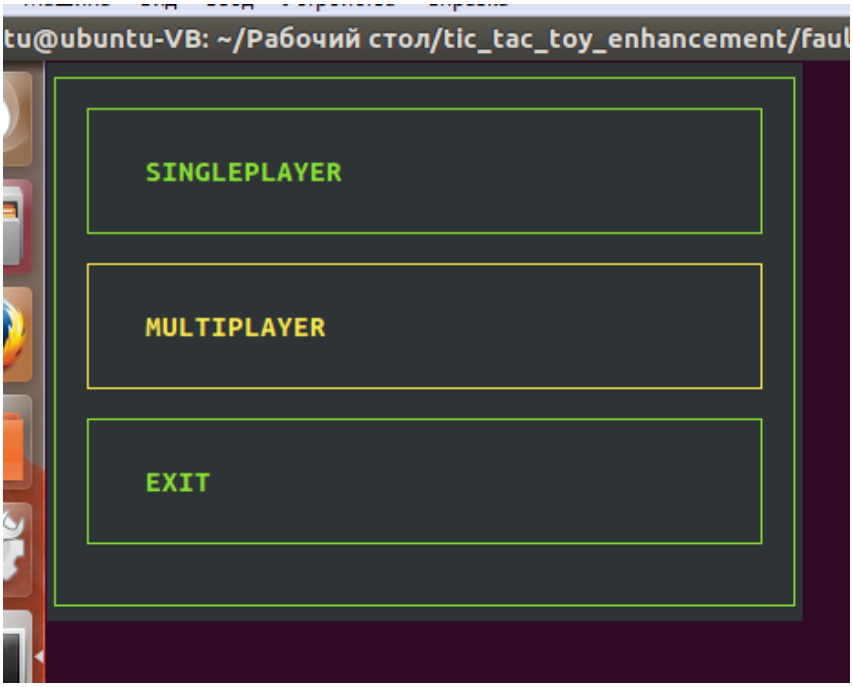

Рисунок 4 – Главное меню

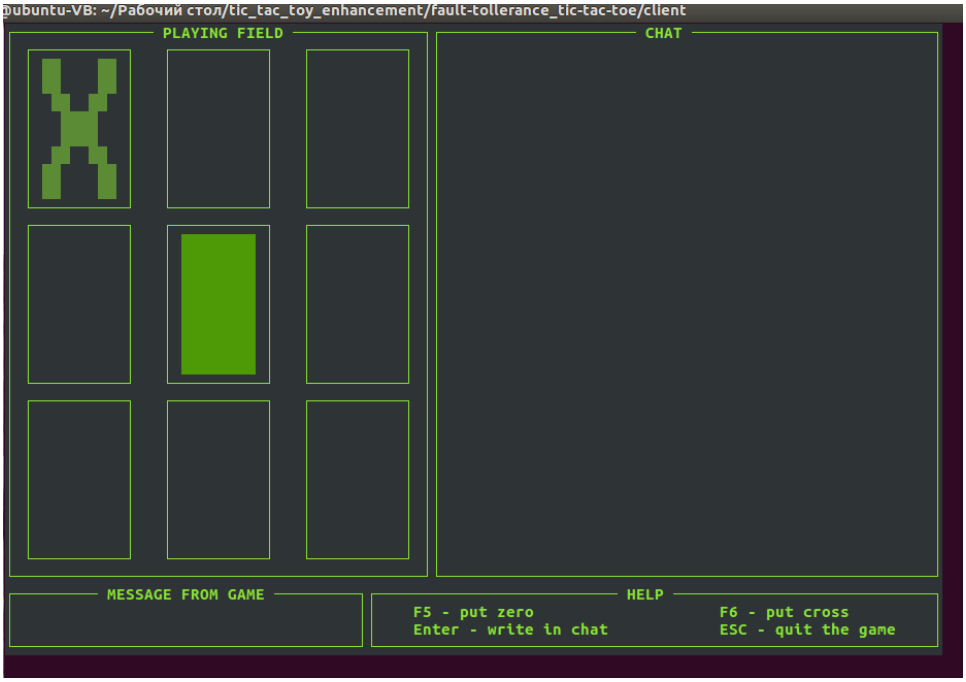

Рисунок 5 – Внешний вид разработанного приложения

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 an-
other...\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 X */
    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 по-
токе)
        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 чте-
ние/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 чте-
ние/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;
}
```