Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных систем и технологий

**Сетевая игра «Крестики-Нолики»**

Курсовая работа

по дисциплине:

«Операционные системы»

Выполнил:

Студент группы 9091

Прокофьев М.Ю.

«\_\_»\_\_\_\_\_.2021г.

Проверил:

Сухарев И.Д.

«\_\_»\_\_\_\_\_.2021г.

Великий Новгород  
2021

**ОГЛАВЛЕНИЕ**

ВВЕДЕНИЕ....................................................................................................................................

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА............................................................................................

1.1 Постановка задачи...................................................................................................................

1.2 Выбор инструментальных средств........................................................................................

1.3 Выбор модели..........................................................................................................................

1.4 Алгоритм решения задачи.......................................................................................................

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА.........................................................................................

2.1 Структура клиента и сервера.................................................................................................

2.2 Программный интерфейс сервера.........................................................................................

2.3 Описание программной реализации клиента.......................................................................

2.4 Системные вызовы..................................................................................................................

2.5 Пример работы приложения...................................................................................................

2.6 Результат работы.....................................................................................................................

ЗАКЛЮЧЕНИЕ..............................................................................................................................

Код из файла server.сpp........................................................................................ ..…………….

Код из файла client.сpp....................................................................................................................

**ВВЕДЕНИЕ**

Цель данной работы – создание консольной версии игры «Крестики-Нолики», а также получение практических навыков разработки и реализации многопользовательских сетевых приложений под управлением операционной системы семейства Unix/Linux.

**Кре́стики-но́лики** — логическая игра между двумя противниками на квадратном поле 3 на 3 клетки или бо́льшего размера. Один из игроков играет «крестиками», второй — «ноликами». Особенностью проектируемой игры заключается в том, что она должна работать под операционной системой Linux.

Правила игры:

В игре участвуют только два игрока.

Игроки по очереди ставят на свободные клетки поля 3х3 знаки (один всегда крестики, другой всегда нолики). Первый, выстроивший в ряд 3 своих фигуры по вертикали, горизонтали или диагонали, выигрывает. Первый ход делает игрок, ставящий крестики.

Обычно по завершении партии выигравшая сторона зачёркивает чертой свои три знака (нолика или крестика), составляющих сплошной ряд.

Функции, реализованные в игре:

1. Подключение двух игроков;
2. Возможность для обоих игроков расставлять крестики или нолики на выделенной для них матрице;
3. Консольный интерфейс для заполненных ячеек в матрице и вывода результата.

**ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА**

**1.1 Постановка задачи**

Задачей данной курсовой работы является разработка сетевой игры «Крестики-Нолики», которая рассчитана на двух игроков. Игра должна работать под управлением операционной системы семейства Unix/Linux. Сетевая часть игры должна быть выполнена на языке C с помощью сокетов предоставляемых системной библиотекой <sys/socket.h>.

**1.2 Выбор инструментальных средств**

Для реализации сетевого взаимодействия используются протокол передачи данных: TCP.

Протокол TCP предоставляет транспортные услуги, отличающиеся от услуг UDP. Вместо ненадежной доставки датаграмм без установления соединений, он обеспечивает гарантированную доставку с установлением соединений между прикладными процессами в виде байтовых потоков.

Протокол TCP используется в тех случаях, когда требуется надежная доставка сообщений. Он освобождает прикладные процессы от необходимости использовать таймауты и повторные передачи для обеспечения надежности.

**1.3 Выбор модели**

Для взаимодействия между игроками выбрана клиент-серверная модель. Сервер обеспечивает хранение всех данных игры.

**1.4 Алгоритм решения задачи**

Подключение игрока/игроков происходит следующим образом:

К серверу подключаются первый и второй игрок, введя адрес сервера и номер порта.

Ход игры:

Сервер ожидает подключения двух клиентов, а затем организует взаимные запросы на клиент и обратно для выбора ведущего игрока из двух и рандомно наделяет каждого из клиентов символом для последующей игры.

**ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА**

**2.1 Структура клиента и сервера**

Соединение между клиентом и сервером. Для игры также используются некоторые дополнительные функции.

**2.2 Программный интерфейс сервера**

Сервер получает от клиента непосредственно саму ячейку для того, чтобы в последствии проверить ее на релевантность хода и отправляет обратно, выдавая информацию о том, какой символ будет записан в выбранную клиентом ячейку матрицы. Возможно и другая вариация исхода, если проверенная ячейка либо совпадет с уже имеющийся, либо какой-то из игроков победит.

**2.3 Описание программной реализации клиента**

При запуске клиента происходит подключение к серверу. У клиента два разных состояния: когда он ходит и когда он ожидает хода другого клиента. Информацию о его состоянии он получает от сервера.

**2.4 Системные вызовы**

**1) socket (int domain, int type, int protocol) -** cлужит для создания сокета в ОС. Имеет три парметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства.

**2) connect(int *sockfd*, const struct sockaddr \**serv\_addr*, socklen\_t *addrlen*) -** служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина.

**3) inet\_pton(int** *af***, const char \****src***, void \****dst***) -**  преобразует строку символов *src* в сетевой адрес (типа *af*), затем копирует полученную структуру с адресом в *dst*. На текущий момент поддерживаются следующие типы адресов: **AF\_INET и AF\_INET6.** Если сетевой адрес был успешно преобразован, то возвращается положительное значение.

**4) read() и write()** - используются для чтения и записи файлов. Могут быть использованы для обмена данных, если в качестве параметров вместо дескрипторов файлов в них задаются дескрипторы сокетов.

**5) bind(int***sockfd***, struct sockaddr** \**my\_addr***, socklen\_t***addrlen***) -** настраивает адрес для созданного сокета. Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес.

**6) listen(int *s*, int *backlog*) -** прослушивает соединения на сокете. Позволяет показать готовность принимать соединения и задать лимит входящих соединений. В качестве первого параметра используется дескриптор сокета. Второй параметр определяет максимальную длину очереди входящих соединений.

**9) accept(int***s***, struct sockaddr** \**addr***, socklen\_t** \**addrlen***) –** 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.

**2.5 Пример работы приложения**

Сначала надо запустить программу сервер, введя в командой строке:

./server

Затем надо запустить программу клиент, для которой формат ввода данных имеет следующий вид:

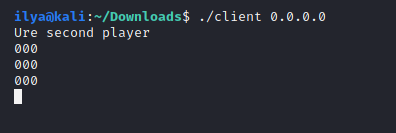
./client [port]

**Режим игрок против игрока**.

Для этого режима программа клиент должен быть запущен два раза.

Клиенты ожидают подключения двух игроков и затем сервер рисует им обнуленную матрицу.

Результат, выведенный на экран 2 игрока:



Результат хода одного из клиентов:



**ЗАКЛЮЧЕНИЕ**

Таким образом, результат курсовой работы соответствует требованиям к реализации. Приложение работает на операционных системах семейства Linux. Для построения сетевой части приложения использован язык C и компилятор gсс. Сетевое взаимодействие было построено с помощью протокола TCP.

Код с комментариями из файла server.сpp

#include <sys/types.h>

#include <sched.h>

#include <pthread.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/wait.h>

int checkTurn(int a[3][3]){

if (a[0][0] == a[1][1] && a[1][1] == a[2][2] && a[0][0] != 0){

return a[1][1];

}

else if (a[0][2] == a[1][1] && a[1][1] == a[2][0] && a[0][2] != 0){

return a[1][3];

}

else if (a[0][0] == a[1][0] && a[1][0] == a[2][0] && a[0][0] != 0){

return a[1][1];

}

else if (a[0][1] == a[1][1] && a[1][1] == a[2][1] && a[0][1] != 0){

return a[1][2];

}

else if (a[0][2] == a[1][2] && a[1][2] == a[2][2] && a[0][2] != 0){

return a[1][3];

}

else if (a[2][0] == a[2][1] && a[2][1] == a[2][2] && a[2][0] != 0){

return a[3][1];

}

else if (a[1][0] == a[1][1] && a[1][1] == a[1][2] && a[1][0] != 0){

return a[2][1];

}

else if (a[0][0] == a[0][1] && a[0][1] == a[0][2] && a[0][0] != 0){

return a[1][1];

}

else{

return 0;

}

}

int main()

{

int firstplayerfd, secondplayerfd, sockfd;

struct sockaddr\_in servaddr;

if ((sockfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0){

perror(NULL);

exit(1);

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(9990);

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

if (bind(sockfd, (struct sockaddr \*) &servaddr, sizeof(servaddr)) < 0){

perror(NULL);

close(sockfd);

exit(1);

}

if (listen(sockfd, 5) < 0){

perror(NULL);

close(sockfd);

exit(1);

}

printf("\nWaiting both players");

pid\_t pid;

pid = fork();

if( pid < 0){

perror("\nChildren below zero");

close(sockfd);

exit(0);

}

else if(pid == 0)

{

while(1){

if((firstplayerfd = accept(sockfd, 0, 0)) < 0){

perror("Client first initializ failed\n");

close(sockfd);

exit(0);

}

if((secondplayerfd = accept(sockfd,0, 0)) < 0){

perror("Client second initializ failed\n");

close(sockfd);

exit(0);

}

pid = fork();

if(pid < 0){

perror("\nChildren bellow zero st2");//step 2

close(sockfd);

exit(0);

}

else if(pid == 0)

{

int a[3][3];

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

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

a[i][j] = 0;

}

}

close(sockfd);

printf("\nSuccess");

int buff = 0;//0 - Стартовый выбор ролей

int x, y; //Элементы матрицы

int callback\_whowins; // Если у нас статус стал 0-м,то выиграл первый сокет && Если 1-ей, тогда победа за 2 сокетом

send(firstplayerfd, &buff, sizeof(int), 0);

buff = 0; send(firstplayerfd, &buff, sizeof(int), 0);

buff = 0; send(secondplayerfd, &buff, sizeof(int), 0);

buff = 1; send(secondplayerfd, &buff, sizeof(int), 0);

recv(firstplayerfd, &buff, sizeof(int), 0);

if(buff == 1)

{

while(1)

{

recv(firstplayerfd, &x, sizeof(int), 0);

recv(firstplayerfd, &y, sizeof(int), 0);

a[x][y] = 5; // крестик

buff = 1;

send(secondplayerfd, &buff, sizeof(int), 0);

send(secondplayerfd, &x, sizeof(int), 0);

send(secondplayerfd, &y, sizeof(int), 0);

if (checkTurn(a) != 0){

callback\_whowins = 0;

break; //+++ закрытие сокетов?///

} //Здесь должна быть проверка матрицы.

buff = 1;

recv(secondplayerfd, &buff, sizeof(buff),0);

recv(secondplayerfd, &x, sizeof(int), 0);

recv(secondplayerfd, &y, sizeof(int), 0);

a[x][y] = 6;//Нолик

send(firstplayerfd, &buff, sizeof(int), 0);

send(firstplayerfd, &x, sizeof(int), 0);

send(firstplayerfd, &y, sizeof(int), 0);

if (checkTurn(a) != 0){

callback\_whowins = 1;

break;

} //Здесь должна быть проверка матрицы.

}

buff = 2;

if (callback\_whowins == 0){

send(firstplayerfd, &buff, sizeof(int), 0);

send(secondplayerfd, &buff, sizeof(int), 0);

buff = 0;

send(firstplayerfd, &buff, sizeof(int), 0);

send(secondplayerfd, &buff, sizeof(int), 0);

}

else if(callback\_whowins == 1){

send(secondplayerfd, &buff, sizeof(int), 0);

send(firstplayerfd, &buff, sizeof(int), 0);

buff = 1;

send(secondplayerfd, &buff, sizeof(int), 0);

send(firstplayerfd, &buff, sizeof(int), 0);

}

else

{

printf("\nChek\_turn() returned unavailable number");

}

}

}

else{

close(firstplayerfd);

close(secondplayerfd);

}

}

close(sockfd);

}

else

{

waitpid(pid,0,0);

}

}

Код из файла client.сpp

#include <sys/types.h>

#include <sys/socket.h>

#include <netinet/in.h>

#include <arpa/inet.h>

#include <string.h>

#include <stdio.h>

#include <errno.h>

#include <unistd.h>

#include <stdlib.h>

#include <stdbool.h>

bool status; // Принимает роль

void Receive(int a[3][3], int sockfd){

// Функция

int buff, x, y;

recv(sockfd, &buff, sizeof(int), 0); //ход сделан// кто-то победил

switch(buff){

case 0: // Назначение ролей

recv(sockfd, &status, sizeof(bool), 0);

break;

case 1:

recv(sockfd, &x, sizeof(int), 0);

recv(sockfd, &y, sizeof(int), 0);

if(status){

a[x][y] = 6;

}

else{

a[x][y] = 5;

}

break;

case 2:

recv(sockfd, &buff, sizeof(int), 0);

if(buff == 0){

printf("\nFirst player win");

}

else{

printf("\nSecond player win");

}

break;

default:

exit(0);

}

}

void SendTurn(int sockfd, int x, int y){

int buff = 1;

write(sockfd, &buff, sizeof(int));

write(sockfd, &x, sizeof(int));

write(sockfd, &y, sizeof(int));

}

void Draw\_map(int a[3][3])

{

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

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

if (a[i][j] == 5){

printf("O");

}

else if(a[i][j] == 6){

printf("X");

}

else{

printf("%i", a[i][j]);

}

}

printf("\n");

}

}

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

typedef int MatrixArray[3];

int sockfd;

struct sockaddr\_in servaddr;

if(argc != 2){

printf("Usage Ip-Adress");

exit(1);

}

if((sockfd = socket(PF\_INET, SOCK\_STREAM, 0)) < 0){

perror("Failed create socket");

close(sockfd);

exit(1);

}

bzero(&servaddr, sizeof(servaddr));

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(9990);

if(inet\_aton(argv[1], &servaddr.sin\_addr) == 0){

printf("Invalid IP adress\n");

close(sockfd);

exit(1);

}

if(connect(sockfd, (struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0){

perror("Connection failed");

close(sockfd);

exit(1);

}

int arrayDef[3][3];

MatrixArray \*a = arrayDef;

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

{

for (int y = 0; y < 3; y++)

{

a[i][y] = 0;

}

}

int x, y;

Receive(a, sockfd);

bool input = false;

if (status)

{

printf("Ure second player\n");

Draw\_map(a);

Receive(a, sockfd);//Первый ход получен!!!

Draw\_map(a);

scanf("%i %i", &x, &y);//Второй ход!!!

if(a[x][y] == 0){

a[x][y] = 6; // крестик

}

else{

while(!input){

printf("Wrnog cell, repeat input \n");

scanf("%d %d", &x, &y);

if(a[x][y] == 0){

a[x][y] == 6;

input = true;

}

}

}

Draw\_map(a);

SendTurn(sockfd, x, y);//Второй ход отправлен!!!

}

else{

printf("Ure first player\n");

input = false;

Draw\_map(a);

scanf("%i %i", &x, &y);//Второй ход!!!

if(a[x][y] == 0)

{

a[x][y] = 5; // Нолик

}

else{

while(!input){

printf("Wrong cell, repeat input \n");

scanf("%d %d", &x, &y);

if(a[x][y] == 0){

a[x][y] == 5;

input = true;

}

}

}

SendTurn(sockfd, x, y);//Первый ход!!!!

Draw\_map(a);

Receive(a, sockfd);

Draw\_map(a);//Нарисовали второй ход

scanf("%i %i", &x, &y);//3 ход!!!

if(a[x][y] == 0){

a[x][y] = 5;

}

Draw\_map(a);//Нарисовали 3 ход

SendTurn(sockfd, x, y);//3 ход!!!!

}

close(sockfd);

}