Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра информационных технологий и систем

**Сетевая игра «Дурак»**

Курсовая работа

по дисциплине:

«Операционные системы»

Разработал:

Студент группы 9091

Флейшер Кевин Освальд

«\_\_».\_\_\_\_\_.2020г.

Проверил:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2021г.

Великий Новгород

2021

Оглавление

ОГЛАВЛЕНИЕ............................................................................................................................... 2

ВВЕДЕНИЕ.................................................................................................................................... 3

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА............................................................................................ 3

1.1 Постановка задачи................................................................................................................... 3

1.2 Выбор инструментальных средств........................................................................................ 3

1.3 Выбор модели.......................................................................................................................... 3

1.4 Алгоритм решения задачи.......................................................................................................3

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА....................................................................................... ..4

2.1 Структура клиента и сервера................................................................................................. 4

2.2 Программный интерфейс сервера......................................................................................... 4

2.3 Описание программной реализации клиента...................................................................... 4

2.4 Системные вызовы................................................................................................................. 4

2.5 Пример работы приложения.................................................................................................. 5

2.6 Результат работы..................................................................................................................... 6

ЗАКЛЮЧЕНИЕ............................................................................................................................. 6

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ...................................................................... 6

Листинг В1 — Код из файла server.h........................................................................................... 7

Листинг В2 — Код из файла client.h........................................................................................... 15

ВВЕДЕНИЕ

Цель данной работы – создание простейшей консольной версии игры «дурак».

Правила игры:

Игрокам раздают по 6 карт, один игрок ходит на другого картой, другой игрок должен отбиться другой картой той же масти и высшего достоинства, либо любой козырной картой(если нападали козырной, то отбиться можно только козырной высшего достоинства), либо карту можно принять, и тогда защищающийся пропускает ход и (в случае с двумя игроками) на него ходят еще раз. Задача игроков избавиться от карт как можно быстрее, единственный игрок имеющий карты считается проигравшим (дураком).

В данной простой реализации в игре участвуют максимум два игрока, нет возможности подкидывать и ходить более чем одной картой.

Функции, реализованные в игре:

1. Подключение двух игроков
2. Консольный интерфейс для взаимодействия игроков

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА

1.1 Постановка задачи

Задачей данной курсовой работы является разработка сетевой игры «Дурак», 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 TCP – сокеты.

1.3 Выбор модели

Для взаимодействия между игроками выбрана клиент-серверная модель. Сервер обеспечивает синхронизацию ходов, хранение колоды, выдачу карт.

1.4 Алгоритм решения задачи

Подключение игрока/игроков происходит следующим образом:

К серверу подключается первый игрок ему присваивается номер 1, ожидает подключение второго. Когда второй подключился, ему присваивается номер 2, после этого оба игрока должны подтвердить готовность к началу игры, после этого игра начинается с хода первого игрока.

Ход игры:

Игрокам показываются разданные им карты и козырь. Начинает ходить первый игрок. Он выбирает карту (путем набирания полного названия этой карты). Ход переходит второму игроку, ему предоставляется выбор, принять карту или отбиваться, если он принимает карту, то первый игрок снова ходит на второго, если он решил отбиваться, то выбирает карту для защиты, если карта удолетворяет всем условиям (наличие данной карты у игрока и ее способностью побить карту, которой походил первый игрок), то второму игроку переходит право ходить(нападать) на первого игрока. И так продолжается, пока у одного из них не кончатся карты.

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА

2.1 Структура клиента и сервера

Соединение между клиентом и сервером происходит в главной функции и одной вспомогательной функцией. Для игры используются отдельные функции и некоторые вспомогательные функции.

2.2 Программный интерфейс сервера

Подключение клиентов и весь процесс игры происходит в главной функции. После подключения запускается бесконечный цикл, где в зависимости от специальных переменных, обеспечивающих синхронизацию, выбирается кто будет ходить и какое действие выполнять(нападать, защищаться). Перед началом цикла эти переменные инициализированы так, что первый игрок нападает, а второй ожидает. Когда первый походил, блок обработки передает второму игроку, какое действие он будет выполнять, а именно защищаться. Первый игрок ожидает. Если второй игрок отбился, то переход хода не происходит, первый игрок продолжает ждать, меняется только переменная, говорящая, что второй игрок теперь нападает, и еще одна переменная уставливается в положение, при котором первый игрок не получает информацию о переходе. Если второй игрок принял карту, то переход хода происходит, но переменные узказывают на то, что первый снова нападает, и после этого второму снова придется защищаться. Во время игры хранение карт и их обработка, сравнения, происходят в виде чисел, т.е. карты с точки зрения сервера – это массив двузначных чисел, первая цифра числа указывает на масть, вторая на достоинство. Карты, которые лежат на руках игроков также хранятся в своих массивах, которые пополняются из основного, меняют свой размер в зависимости от ситуации. В начале игры массив карт перемешивается и заполняются массивы карт каждого игрока и они отправляются игрокам.

2.3 Описание программной реализации клиента

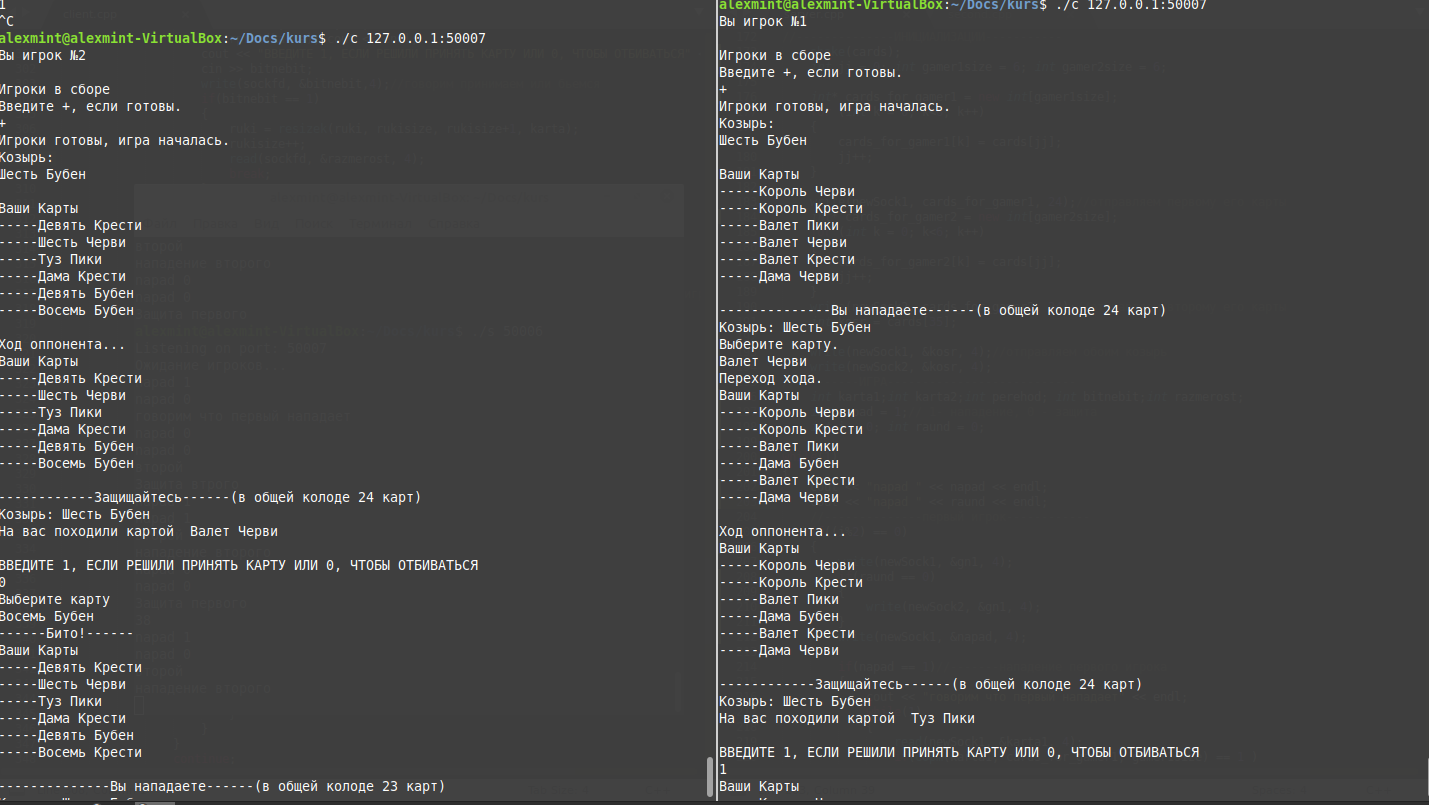
При запуске клиента происходит подключение к серверу. Клиенту приходит информация о том, какой у него номер и происходит ожидание второго игрока. После сбора игроков, они подтвержают начало игры. С сервера приходит число, показывающее чей сейчас ход, если ход равен номеру игрока, то этот игрок ходит и ему приходит число, определяющее его действие – защита либо нападение. Другой игрок, у которого его номер и полученное число не совпадают, попадает в ожидание окончания хода. Вся обработка карт происходит в виде цифр, но имеются функции, которые при отображении на экран конвертируют в текстовый вид название карты, а при вводе пользователем названия, конвертирует обратно в числа и отправляют на сервер.

2.4 Системные вызовы

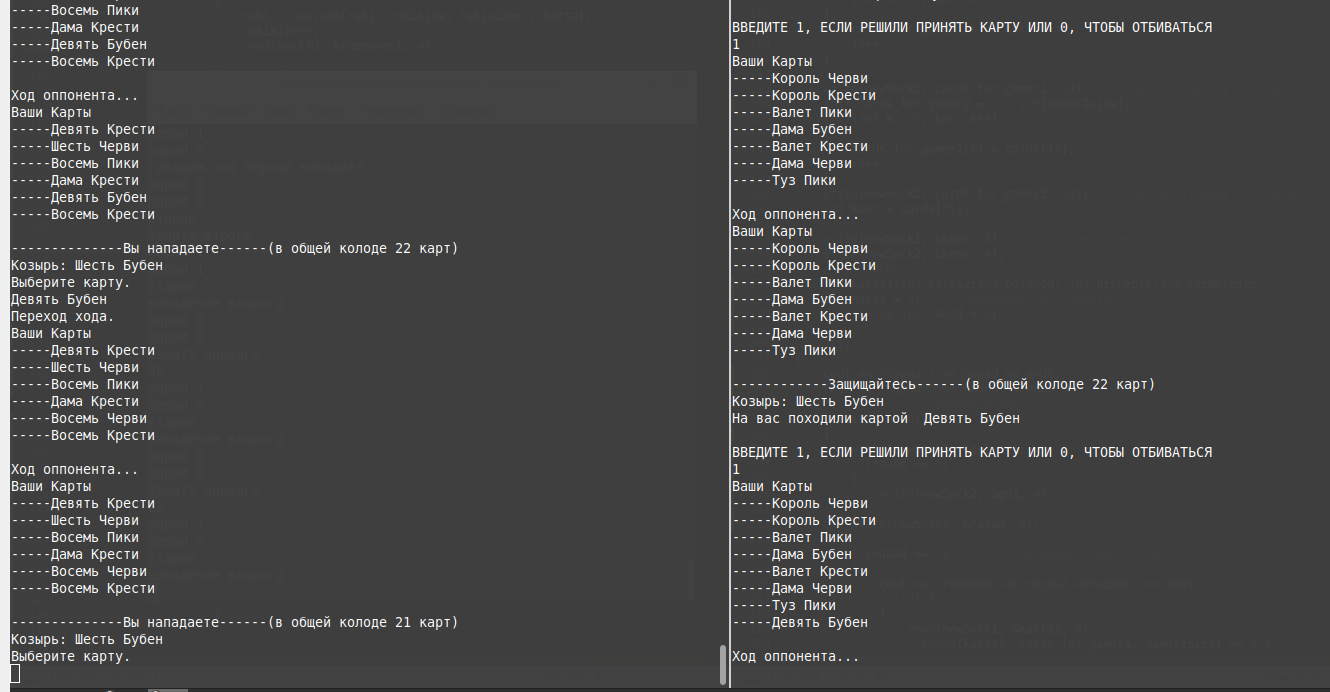
|  |  |
| --- | --- |
| socket() | Служит для создания сокета в ОС. Имеет три парметра. Первый указывает, к какому семейству протоколов относится создаваемый сокет, а второй и третий параметры определяют конкретный протокол внутри данного семейства. |
| Connect() | Служит для установки логического соединения со стороны клиента. Имеет три параметра: дескриптор активного сокета, через который будет устанавливаться соединение, полный адрес сокета сервера и его длина. |
| 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. |
| bind() | Настраивает адрес для созданного сокета. Первый параметр вызова должен содержать дескриптор сокета, для которого производится настройка адреса. Второй и третий параметры задают этот адрес. |
| listen() | 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. |

2.5 Пример работы приложения

Подключается первый игрок, видит свой номер, подключается второй игрок, видит свой номер, и сразу оба видят надпись, что оба в сборе, они оба пишут и отправляют плюс и игра начинается:

В этом случае первый игрок нападает, второй защищается и успешно отбивает карту и сам нападает

Далее, когда второй игрок напал, первый игрок принимает карту и она появляется в его массиве, второй игрок снова нападет, и первый снова принимает, и уже имеет 8 карт, второй в этот момент имеет 6 карт и пополняет их из основной колоды.



2.6 Результат работы

Создана работающая игра сетевая игра крестики-ноли, в которой для сетевого взаимодействия игроков используются клиент-серверная модель, а обмен данными осуществляется через TCP – сокеты.

ЗАКЛЮЧЕНИЕ

В результате, было создано серверное приложение соответствующее цели работы. Приложение работает на операционных системах семейства Linux. Для построения сетевой части приложения использован язык C++ и компилятор g++. Сетевое взаимодействие было построено с помощью протокола TCP.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Скотт Граннеман «Linux. Карманный справочник»

2. Брюс Моли «Unix/Linux. Теория и практика программирования»

3. <https://linux.die.net/>

4. https://ru.wikipedia.org/wiki/Дурак\_(карточная\_игра)

[Листинг В1 — Код из файла server.h](#__RefHeading___Toc10981_812659571)

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <stdlib.h> |
|  | #include <time.h> |
|  | #include <sys/types.h> |
|  | #include <sys/wait.h> |
|  | #include <sys/socket.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <netinet/in.h> |
|  | #include <arpa/inet.h> |
|  | #include <unistd.h> |
|  | #include <iostream> |
|  | #include <time.h> |
|  |  |
|  | using namespace std; |
|  |  |
|  |  |
|  | void printmas(int\* mas, int s) |
|  | { |
|  | for (int i = 0; i < s; i++) |
|  | { |
|  | cout << mas[i] << " "; |
|  | } |
|  | cout << endl; |
|  | } |
|  |  |
|  | void shake(int\* c) |
|  | { |
|  | int ind; int temp; |
|  | srand(time(NULL)); |
|  | for(int i = 0; i < 36; i++) |
|  | { |
|  | ind = rand()%(36-i) + i; |
|  | temp = c[i]; |
|  | c[i] = c[ind]; |
|  | c[ind] = temp; |
|  | } |
|  | } |
|  |  |
|  | bool copm(int kart, int\* ruki, int s) |
|  | { |
|  | for(int i = 0; i < s; i++) |
|  | { |
|  | if( ruki[i] == kart ) |
|  | { |
|  | return 0; |
|  | } |
|  | } |
|  | return 1; |
|  | } |
|  |  |
|  | int\* resizek(int \*mas, int size, int new\_size, int karta = 0) |
|  | { |
|  | int\*mm = new int[new\_size]; int index = 0; |
|  | if(size > new\_size) |
|  | { |
|  | for(int i = 0; i < size; i++) |
|  | { |
|  | if(mas[index] == karta) |
|  | { |
|  | break; |
|  | } |
|  | index++; |
|  | } |
|  | int tmp = mas[size-1]; |
|  | mas[size-1] = mas[index]; |
|  | mas[index] = tmp; |
|  | } |
|  | for(int p = 0; p < new\_size; p++) |
|  | { |
|  | if (p==size) |
|  | { |
|  | mm[p]=karta; |
|  | } |
|  | else |
|  | { |
|  | mm[p]=mas[p]; |
|  | } |
|  | } |
|  | delete []mas; |
|  | return mm; |
|  | } |
|  |  |
|  | void rukiupdate(int\* ruki, int karta, int zamena, int size) |
|  | { |
|  | for(int i = 0; i < size; i++) |
|  | { |
|  | if(ruki[i] == karta) |
|  | { |
|  | ruki[i] = zamena; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | bool bitcheck(int k1, int k2, int kosr) //(защита, нападение) |
|  | { |
|  | if((k1/10 == kosr/10) && (k2/10 != kosr/10)) |
|  | { return 1;} |
|  | else if((k1/10 != kosr/10) && (k2/10 == kosr/10)) |
|  | {return 0;} |
|  | else if(k1/10 != k2/10) |
|  | {return 0;} |
|  | else |
|  | { |
|  | if(k1%10 > k2%10) |
|  | {return 1;} |
|  | else |
|  | {return 0;} |
|  | } |
|  |  |
|  | } |
|  |  |
|  | int main(int argc, char\* argv[]) |
|  | { |
|  | setlocale(LC\_ALL, "Russian"); |
|  |  |
|  | int port = atoi(argv[1]); |
|  | struct sockaddr\_in server\_addr; |
|  |  |
|  | bzero(&server\_addr, sizeof(server\_addr)); |
|  | server\_addr.sin\_family = PF\_INET; |
|  | server\_addr.sin\_addr.s\_addr = htonl(INADDR\_ANY); |
|  | server\_addr.sin\_port = htons(port); |
|  |  |
|  | int sock = socket(PF\_INET, SOCK\_STREAM, 0); |
|  | if (sock < 0) |
|  | { |
|  | perror("socket() error"); |
|  | exit(-1); |
|  | } |
|  | if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) |
|  | { |
|  | server\_addr.sin\_port = htons(port+1);; |
|  | if (bind(sock, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) < 0) |
|  | { |
|  | perror("bind() error"); |
|  | exit(-1); |
|  | } |
|  | } |
|  | listen(sock, 2); |
|  | cout << "Listening on port: " << ntohs(server\_addr.sin\_port) << endl; |
|  | int cards[36] |
|  | { |
|  | 10,11,12,13,14,15,16,17,18, |
|  | 20,21,22,23,24,25,26,27,28, |
|  | 30,31,32,33,34,35,36,37,38, |
|  | 40,41,42,43,44,45,46,47,48 |
|  | }; |
|  | shake(cards); |
|  |  |
|  | cout << "Ожидание игроков..." << endl; |
|  | //---------ОЖИДАНИЕ ИГРОКОВ------------- |
|  | int newSock1 = accept(sock, NULL, NULL); |
|  | int gn1 = 1; |
|  | write(newSock1, &gn1, 4);//отправить первому игроку его номер |
|  | int newSock2 = accept(sock, NULL, NULL); |
|  | int gn2 = 2; |
|  | write(newSock2, &gn2, 4); //отправка второму игроку его номер |
|  | char con = 0; |
|  | write(newSock1, &con, 1);//говорим игрокам, что все вошли |
|  | write(newSock2, &con, 1); |
|  | char rdy1,rdy2; |
|  | read(newSock1, &rdy1, 4);//ожидаем готовность игроков |
|  | read(newSock2, &rdy2, 4); |
|  | if(rdy1 == '+' && rdy2 == '+') |
|  | { |
|  | write(newSock1, &con, 1);//говорим всем, что все готовы |
|  | write(newSock2, &con, 1); |
|  | } |
|  | //--------------ИНИЦИАЛИЗАЦИИ----------- |
|  | shake(cards); |
|  | int jj = 0; int gamer1size = 6; int gamer2size = 6; |
|  |  |
|  | int\* cards\_for\_gamer1 = new int[gamer1size]; |
|  | for (int k = 0; k<6; k++) |
|  | { |
|  | cards\_for\_gamer1[k] = cards[jj]; |
|  | jj++; |
|  | } |
|  |  |
|  | write(newSock1, cards\_for\_gamer1, 24);//отправляем первому его карты |
|  | int\* cards\_for\_gamer2 = new int[gamer2size]; |
|  | for (int k = 0; k<6; k++) |
|  | { |
|  | cards\_for\_gamer2[k] = cards[jj]; |
|  | jj++; |
|  | } |
|  | write(newSock2, cards\_for\_gamer2, 24);//отправляем второму его карты |
|  | int kosr = cards[35]; |
|  |  |
|  | write(newSock1, &kosr, 4);//отправляем обоим козырь |
|  | write(newSock2, &kosr, 4); |
|  | //---------ИГРА--------------------------- |
|  | int karta1;int karta2;int perehod; int bitnebit;int razmerost; |
|  | int napad = 1;// 1- нападение, 0 - защита |
|  | int i = 0; int raund = 0; |
|  |  |
|  | while(1) |
|  | { |
|  | //cout << "napad " << napad << endl; |
|  | //cout << "raund " << raund << endl; |
|  | //--------------первый игрок------------ |
|  | if((i%2) == 0) |
|  | { |
|  | write(newSock1, &gn1, 4); |
|  | if(raund == 0) |
|  | { |
|  | write(newSock2, &gn1, 4); |
|  | } |
|  | write(newSock1, &napad, 4); |
|  |  |
|  | if(napad == 1)//-------нападение первого игрока |
|  | { |
|  | //cout << "говорим что первый нападает" << endl; |
|  | while(1) |
|  | { |
|  | read(newSock1, &karta1, 4); |
|  | if(copm(karta1, cards\_for\_gamer1, gamer1size) == 1 ) |
|  | { |
|  | perehod = 0; |
|  | write(newSock1, &perehod, 4);//сообщаем, что ход неудачный |
|  | continue; |
|  | } |
|  | else |
|  | { |
|  | perehod = 1; |
|  | write(newSock1, &perehod, 4);//сообщаем, что ход успешный |
|  | razmerost = 36 - jj ; |
|  | write(newSock1, &razmerost, 4);//передаем остаток в колоде |
|  | //изменение в картах на руках |
|  | if(gamer1size==1 && razmerost==0) |
|  | { |
|  | napad = 3; |
|  | write(newSock2, &napad, 4); |
|  | } |
|  | else if(gamer1size < 7 && razmerost > 1) |
|  | { |
|  | write(newSock1, &cards[jj], 4); |
|  | rukiupdate(cards\_for\_gamer1, karta1 , cards[jj], gamer1size); |
|  | jj++; |
|  | } |
|  | else if(gamer1size < 7 && razmerost == 1) |
|  | { |
|  | write(newSock1, &kosr, 4); jj++; |
|  | } |
|  | else if(gamer1size < 7 && razmerost == 0) |
|  | { |
|  | cards\_for\_gamer1 = resizek(cards\_for\_gamer1, gamer1size, gamer1size-1); |
|  | gamer1size--; |
|  | } |
|  | napad = 0; |
|  | i++; |
|  | raund = 0; |
|  | write(newSock2, &napad, 4); |
|  | break; |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | if(napad == 0)//защита первого игрока |
|  | { |
|  | //cout << "Защита первого" << endl; |
|  | while(1) |
|  | { |
|  | write(newSock1, &karta1, 4); |
|  | read(newSock1, &bitnebit, 4); |
|  | if (bitnebit) |
|  | { |
|  | cards\_for\_gamer1 = resizek(cards\_for\_gamer1, gamer1size, gamer1size+1, karta1); |
|  | gamer1size++; |
|  | napad = 1; |
|  | write(newSock1, &razmerost, 4); |
|  | write(newSock2, &napad, 4); |
|  | i++; |
|  | break; |
|  | } |
|  | else if(!bitnebit) |
|  | { |
|  | read(newSock1, &karta2, 4); |
|  | if(copm(karta2, cards\_for\_gamer1, gamer1size)) |
|  | { |
|  | perehod = 0; |
|  | write(newSock1, &perehod, 4); |
|  | continue; |
|  | } |
|  | else if(!copm(karta2, cards\_for\_gamer1, gamer1size)) |
|  | { |
|  | if (bitcheck(karta2, karta1, kosr)) |
|  | { |
|  | perehod = 1; |
|  | write(newSock1, &perehod, 4); |
|  | //------------------------- |
|  | razmerost = 36 - jj; |
|  | write(newSock1, &razmerost, 4);//передаем остаток в колоде |
|  | //изменение в картах на руках |
|  | if(gamer1size==1 && razmerost==0) |
|  | { |
|  | napad = 3; |
|  | write(newSock2, &napad, 4); |
|  | } |
|  | if(gamer1size < 7 && razmerost > 1) |
|  | { |
|  | write(newSock1, &cards[jj], 4); |
|  | rukiupdate(cards\_for\_gamer1, karta2 , cards[jj], gamer1size); |
|  | jj++; |
|  | } |
|  | if(gamer1size < 7 && razmerost == 1) |
|  | { |
|  | write(newSock1, &kosr, 4); jj++; |
|  | } |
|  | if(gamer1size < 7 && razmerost == 0) |
|  | { |
|  | cards\_for\_gamer1 = resizek(cards\_for\_gamer1, gamer1size, gamer1size-1); |
|  | gamer1size--; |
|  | } |
|  | //------------------------- |
|  | napad = 1; raund= 1; |
|  | write(newSock1, &napad, 4); |
|  | break; |
|  | }else |
|  | { |
|  | perehod = 0; |
|  | write(newSock1, &perehod, 4); |
|  | continue; |
|  | } |
|  | } |
|  | } |
|  | else |
|  | { |
|  | continue; |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | } |
|  | else |
|  | { |
|  | // cout << "второй" << endl; |
|  | if(raund == 0) |
|  | { |
|  | write(newSock1, &gn2, 4); |
|  | } |
|  | write(newSock2, &gn2, 4); |
|  | write(newSock2, &napad, 4); |
|  | if(napad == 1) //нападение второго игрока |
|  | { |
|  | //cout <<"нападение второго" << endl; |
|  | while(1) |
|  | { |
|  | read(newSock2, &karta1, 4); |
|  | if(copm(karta1, cards\_for\_gamer2, gamer2size)) |
|  | { |
|  | perehod = 0; |
|  | write(newSock2, &perehod, 4);//сообщаем, что ход неудачный |
|  | continue; |
|  | } |
|  | else if(!copm(karta1, cards\_for\_gamer2, gamer2size)) |
|  | { |
|  | perehod = 1; |
|  | write(newSock2, &perehod, 4);//сообщаем, что ход успешный |
|  | razmerost = 36 - jj; |
|  | write(newSock2, &razmerost, 4);//передаем остаток в колоде |
|  | if(gamer2size==0 && razmerost==0) |
|  | { |
|  | napad = 3; |
|  | write(newSock1, &napad, 4); |
|  | } |
|  | if(gamer2size < 7 && razmerost > 1) |
|  | { |
|  | write(newSock2, &cards[jj], 4); |
|  | rukiupdate(cards\_for\_gamer2, karta1 , cards[jj], gamer2size); |
|  | jj++; |
|  | } |
|  | if(gamer2size < 7 && razmerost == 1) |
|  | { |
|  | write(newSock2, &kosr, 4); jj++; |
|  | } |
|  | if(gamer2size < 7 && razmerost == 0) |
|  | { |
|  | cards\_for\_gamer2 = resizek(cards\_for\_gamer2, gamer2size, gamer2size-1); |
|  | gamer2size--; |
|  | } |
|  | napad = 0; |
|  | i++; |
|  | raund = 0; |
|  | write(newSock1, &napad, 4); |
|  | break; |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | if(napad == 0) //защита второго игрока |
|  | { |
|  | //cout << "Защита втрого"<< endl; |
|  | while(1) |
|  | { |
|  | write(newSock2, &karta1, 4);//говорим другому чем на него походили |
|  | read(newSock2, &bitnebit, 4); |
|  |  |
|  | if (bitnebit == 1) |
|  | { |
|  | cards\_for\_gamer2 = resizek(cards\_for\_gamer2, gamer2size, gamer2size+1, karta1); |
|  | gamer2size++; |
|  | napad = 1; |
|  | write(newSock2, &razmerost, 4); |
|  | write(newSock1, &napad, 4); |
|  | i++; |
|  | break; |
|  | }else if(!bitnebit) |
|  | { |
|  | read(newSock2, &karta2, 4); |
|  | if(copm(karta2, cards\_for\_gamer2, gamer2size)) |
|  | { |
|  | perehod = 0; |
|  | write(newSock2, &perehod, 4); |
|  | continue; |
|  | } |
|  | else if(!copm(karta2, cards\_for\_gamer2, gamer2size)) |
|  | { |
|  | if (bitcheck(karta2, karta1, kosr)) |
|  | { |
|  | perehod = 1; |
|  | write(newSock2, &perehod, 4); |
|  | //------------------------- |
|  | razmerost = 36 - jj; |
|  | write(newSock2, &razmerost, 4);//передаем остаток в колоде |
|  | //изменение в картах на руках |
|  | if(gamer2size==1 && razmerost==0) |
|  | { |
|  | napad = 3; |
|  | write(newSock2, &napad, 4); |
|  | } |
|  | if(gamer2size < 7 && razmerost > 1) |
|  | { |
|  | write(newSock2, &cards[jj], 4); |
|  | rukiupdate(cards\_for\_gamer2, karta2 , cards[jj], gamer2size); |
|  | jj++; |
|  | } |
|  | if(gamer2size < 7 && razmerost == 1) |
|  | { |
|  | write(newSock2, &kosr, 4); jj++; |
|  | rukiupdate(cards\_for\_gamer2, karta1 , kosr, gamer2size); |
|  | } |
|  | if(gamer2size < 7 && razmerost == 0) |
|  | { |
|  | cards\_for\_gamer2 = resizek(cards\_for\_gamer2, gamer2size, gamer2size-1); |
|  | gamer2size--; |
|  | } |
|  | napad = 1; |
|  | raund++; |
|  | break; |
|  | }else |
|  | { |
|  | perehod = 0; |
|  | write(newSock2, &perehod, 4); |
|  | continue; |
|  | } |
|  | } |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | } |
|  | } |
|  | return 0; |
|  | } |

[Листинг В2 — Код из файла client.h](#__RefHeading___Toc10983_812659571)

|  |
| --- |
| #include <stdio.h> |
|  | #include <string.h> |
|  | #include <stdlib.h> |
|  | #include <time.h> |
|  | #include <sys/types.h> |
|  | #include <sys/socket.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <netinet/in.h> |
|  | #include <arpa/inet.h> |
|  | #include <unistd.h> |
|  | #include <iostream> |
|  |  |
|  | using namespace std; |
|  |  |
|  | void rukiupdate(int\* ruki, int karta, int zamena, int size) |
|  | { |
|  | for(int i = 0; i < size; i++) |
|  | { |
|  | if(ruki[i] == karta) |
|  | { |
|  | ruki[i] = zamena; |
|  | break; |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void interpret(int k) |
|  | { |
|  | if(k%10 == 0) |
|  | {cout << "Шесть ";} |
|  | else if(k%10 == 1) |
|  | {cout << "Семь "; } |
|  | else if(k%10 == 2) |
|  | {cout << "Восемь ";} |
|  | else if(k%10 == 3) |
|  | {cout << "Девять ";} |
|  | else if(k%10 == 4) |
|  | {cout << "Десять ";} |
|  | else if(k%10 == 5) |
|  | {cout << "Валет "; } |
|  | else if(k%10 == 6) |
|  | {cout << "Дама ";} |
|  | else if(k%10 == 7) |
|  | {cout << "Король ";} |
|  | else if(k%10 == 8) |
|  | {cout << "Туз ";} |
|  |  |
|  | if(k/10 == 1) |
|  | {cout << "Черви ";} |
|  | else if(k/10 == 2) |
|  | {cout << "Бубен ";} |
|  | else if(k/10 == 3) |
|  | {cout << "Пики ";} |
|  | else if(k/10 == 4) |
|  | {cout << "Крести ";} |
|  | cout << endl; |
|  | } |
|  |  |
|  | void printmas(int\* mas, int size) |
|  | { |
|  | for (int i = 0; i < size; i++) |
|  | { |
|  | cout << "-----"; |
|  | interpret(mas[i]); |
|  | } |
|  | cout << endl; |
|  | } |
|  |  |
|  | int trans(char\* line, char\* line2) |
|  | { |
|  | char transed[2]; |
|  |  |
|  | if(!strncmp(line, "Шесть", 5)) |
|  | {transed[1] = '0';} |
|  | else if(!strncmp(line, "Семь", 4)) |
|  | {transed[1] = '1';} |
|  | else if(!strncmp(line, "Восемь", 6)) |
|  | {transed[1] = '2';} |
|  | else if(!strncmp(line, "Девять", 6)) |
|  | {transed[1] = '3';} |
|  | else if(!strncmp(line, "Десять", 6)) |
|  | {transed[1] = '4';} |
|  | else if(!strncmp(line, "Валет", 5)) |
|  | {transed[1] = '5';} |
|  | else if(!strncmp(line, "Дама", 4)) |
|  | {transed[1] = '6';} |
|  | else if(!strncmp(line, "Король", 6)) |
|  | {transed[1] = '7';} |
|  | else if(!strncmp(line, "Туз", 3)) |
|  | {transed[1] = '8';} |
|  |  |
|  | if(!strncmp(line2, "Черви", 5)) |
|  | {transed[0] = '1';} |
|  | else if(!strncmp(line2, "Бубен", 5)) |
|  | {transed[0] = '2';} |
|  | else if(!strncmp(line2, "Пики", 4)) |
|  | {transed[0] = '3';} |
|  | else if(!strncmp(line2, "Крести", 6)) |
|  | {transed[0] = '4';} |
|  |  |
|  | return atoi(transed); |
|  | } |
|  |  |
|  | int ipget(char\* arg, char\* ip) |
|  | { |
|  | int iplen; |
|  | int i = 0; |
|  | while(arg) |
|  | { |
|  | if(arg[i] == ':') |
|  | { |
|  | iplen = i; |
|  | break; |
|  | } |
|  | i++; |
|  | } |
|  | i = 0; |
|  | for (char\* l = ip; l < ip + iplen ; l++) |
|  | { |
|  | \*l = \*(arg + i); |
|  | i++; |
|  | } |
|  | \*(ip + i) = '\0'; |
|  | return iplen; |
|  | } |
|  |  |
|  | int\* resizek(int \*mas, int size, int new\_size, int karta = 0) |
|  | { |
|  | int\*mm = new int[new\_size]; int index = 0; |
|  |  |
|  | if(size > new\_size) |
|  | { |
|  | for(int i = 0; i < size; i++) |
|  | { |
|  | if(mas[index] == karta) |
|  | { |
|  | break; |
|  | } |
|  | index++; |
|  | } |
|  | int tmp = mas[size-1]; |
|  | mas[size-1] = mas[index]; |
|  | mas[index] = tmp; |
|  | } |
|  |  |
|  | for(int p = 0; p < new\_size; p++) |
|  | { |
|  | if (p==size) |
|  | { |
|  | mm[p]=karta; |
|  | } |
|  | else |
|  | { |
|  | mm[p]=mas[p]; |
|  | } |
|  | } |
|  | delete []mas; |
|  | return mm; |
|  |  |
|  | } |
|  |  |
|  | int main(int argc, char\* argv[]) |
|  | { |
|  | setlocale(LC\_ALL, "Russian"); |
|  | if(argc != 2 ) |
|  | {cout << "Error: program needs socket." << endl;} |
|  | int ipl; char ip[15]; |
|  | ipl = ipget(argv[1], ip); |
|  | char\* prt = new char[strlen(argv[1]) - ipl + 1]; |
|  | prt = argv[1] + ipl+1; |
|  |  |
|  | struct sockaddr\_in server\_addr; |
|  | bzero(&server\_addr, sizeof(server\_addr)); |
|  | server\_addr.sin\_family = PF\_INET; |
|  | server\_addr.sin\_port = htons(atoi(prt)); |
|  | inet\_aton(ip, &server\_addr.sin\_addr); |
|  |  |
|  | int sockfd = socket(PF\_INET, SOCK\_STREAM, 0); |
|  | if (sockfd < 0) |
|  | { |
|  | perror("socket() error"); |
|  | exit(-1); |
|  | } |
|  |  |
|  | if(connect(sockfd, (struct sockaddr\*) &server\_addr, sizeof(server\_addr)) < 0) |
|  | { |
|  | perror("connect() error"); |
|  | close(sockfd); |
|  | exit(-1); |
|  | } |
|  | //-----СБОР ИГРОКОВ-------- |
|  | int n; |
|  | read(sockfd, &n, 4);//номер игрока |
|  | cout << "Вы игрок №" << n << endl << endl; |
|  | char con = 1; |
|  | read(sockfd, &con, 1);//ожидание подключения второго игрока |
|  | if(!con) |
|  | { |
|  | cout << "Игроки в сборе" << endl; |
|  | } |
|  | cout << "Введите +, если готовы." << endl; |
|  | char rdy; |
|  | while(1) |
|  | { |
|  | cin >> rdy; |
|  | if (rdy == '+') |
|  | { |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | cout << "Это не плюс." << endl; |
|  | continue; |
|  | } |
|  | } |
|  | write(sockfd, &rdy, 1);//отправка подтверждения о готовности |
|  | read(sockfd, &con, 1);//сигнал о готовности обоих игроков |
|  | if(!con) |
|  | {cout << "Игроки готовы, игра началась." << endl;} |
|  | //--------ИНИЦИАЛИЗАЦИИ---------------- |
|  | int rukisize = 6; |
|  | int\* ruki = new int[rukisize]; |
|  | int kosr; |
|  | read(sockfd, ruki, 24);//прием выданных начальных карт |
|  | read(sockfd, &kosr, 4);//прием инфы о козыре |
|  | cout << "Козырь: " <<endl; |
|  | interpret(kosr); |
|  | cout << endl; |
|  |  |
|  |  |
|  | int hod;char line[32]; int karta; |
|  | char line2[32]; int perehod; |
|  | int napad = 1;int karta2; int bitnebit; |
|  | int vzialkartu; int razmerost = 24; |
|  |  |
|  | //-----------ИГРА-------------- |
|  | while(1) |
|  | { |
|  | read(sockfd, &hod, 4);//инфа о том, чей сейчас ход |
|  | cout << "Ваши Карты" << endl; |
|  | printmas(ruki, rukisize); |
|  | //cout << "hod = "<< hod << endl; |
|  | if (hod == n) |
|  | { |
|  | read(sockfd, &napad, 4); |
|  | if(napad == 1)//нападение |
|  | { |
|  | while(1) |
|  | { |
|  | cout << "--------------Вы нападаете------(в общей колоде " << razmerost<< " карт)"<< endl; |
|  | cout << "Козырь: " ; |
|  | interpret(kosr); |
|  | cout << "Выберите карту." << endl; |
|  | cin >> line >> line2; |
|  | karta = trans(line, line2); |
|  | write(sockfd, &karta, 4);//отправка выбранной карты |
|  | read(sockfd, &perehod, 4);//инфа о том, есть ли выбраная карта у игрока |
|  | if (perehod) |
|  | { |
|  | read(sockfd, &razmerost, 4); |
|  | if(rukisize == 1 && razmerost == 0) |
|  | { |
|  | cout << "Поздравляем, вы победили!" << endl; |
|  | close(sockfd); |
|  | exit(0); |
|  | } |
|  | else if(rukisize < 7 && razmerost!=0) |
|  | { |
|  | read(sockfd, &vzialkartu, 4); |
|  | rukiupdate(ruki, karta , vzialkartu,rukisize); |
|  | } |
|  | else if((rukisize < 7 && razmerost == 0) || rukisize > 6) |
|  | { |
|  | ruki = resizek(ruki, rukisize, rukisize-1); |
|  | rukisize--; |
|  |  |
|  | } |
|  | cout << "Переход хода." << endl; |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | cout << "У вас нет такой карты!" << endl; |
|  | continue; |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | if(napad == 0)//защита |
|  | { |
|  |  |
|  | while(1) |
|  | { |
|  | cout << "------------Защищайтесь------(в общей колоде " << razmerost<< " карт)"<< endl; |
|  | cout << "Козырь: "; |
|  | interpret(kosr); |
|  | cout << "На вас походили картой "; |
|  | read(sockfd, &karta, 4); |
|  | interpret(karta); cout << endl; |
|  | cout << "ВВЕДИТЕ 1, ЕСЛИ РЕШИЛИ ПРИНЯТЬ КАРТУ ИЛИ 0, ЧТОБЫ ОТБИВАТЬСЯ" << endl; |
|  | cin >> bitnebit; |
|  | write(sockfd, &bitnebit,4);//говорим принимаем или бьемся |
|  | if(bitnebit == 1) |
|  | { |
|  | ruki = resizek(ruki, rukisize, rukisize+1, karta); |
|  | rukisize++; |
|  | read(sockfd, &razmerost, 4); |
|  | break; |
|  | } |
|  | else if(!bitnebit) |
|  | { |
|  | cout << "Выберите карту" << endl; |
|  | cin >> line >> line2; |
|  | karta2 = trans(line, line2); |
|  | write(sockfd, &karta2, 4);//отправка выбранной карты |
|  | read(sockfd, &perehod, 4);//инфа о том, есть ли выбраная карта у игрока |
|  | if (perehod) |
|  | { |
|  | read(sockfd, &razmerost, 4); |
|  | if(rukisize == 1 && razmerost == 0) |
|  | { |
|  | cout << "Поздравляем, вы победили!" << endl; |
|  | close(sockfd); |
|  | exit(0); |
|  | } |
|  | if(rukisize < 7 && razmerost!=0) |
|  | { |
|  | read(sockfd, &vzialkartu, 4); |
|  | rukiupdate(ruki, karta2 , vzialkartu, rukisize); |
|  | } |
|  |  |
|  | if((rukisize < 7 && razmerost == 0) || rukisize > 6) |
|  | { |
|  | ruki = resizek(ruki, rukisize, rukisize-1); |
|  | rukisize--; |
|  | } |
|  | cout << "------Бито!------" << endl; |
|  | break; |
|  | } |
|  | else |
|  | { |
|  | cout << "У вас нет такой карты!" << endl; |
|  | continue; |
|  | } |
|  | } |
|  | } |
|  | continue; |
|  | } |
|  | } |
|  | else |
|  | { |
|  | cout << "Ход оппонента..." << endl; |
|  | read(sockfd, &napad, 4); |
|  | if(napad == 3) |
|  | { |
|  | cout << "Игра окончена. Вы дурак." << endl; |
|  | close(sockfd); |
|  | exit(0); |
|  | } |
|  | } |
|  | } |
|  | return 0; |
|  | } |