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 «Соревновательная математика»**

Курсовая работа

по дисциплине:

«Операционные системы»

Разработал:

Студент группы 9091

Кудрявцев В.В.

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

«\_\_».\_\_\_\_\_.2021г.

Проверил:

Ананьев В. В.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2021г.

**Великий Новгород**

**2021**

ВВЕДЕНИЕ

Целью данной работы является создание сетевой игры «Соревновательная математика». Это игра в нарисованные слова и шарады.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux.

**Суть игры, следующая**:

1. *Игрокам даётся одинаковое случайное математическое выражение.*
2. *Игрокам необходимо дать ответ как можно быстрее.*
3. *Выигрывает только один игрок – тот, кто ответил правильно и быстрее всех.*
4. *Игра длится до заданного количества победных очков*.

Игра должна быть сетевой и обслуживать от 1 до 5 игроков. Так же в программной реализации должно быть разделение глобальных подзадач на отдельные потоки.

Я считаю выбранную тему довольно актуальной и интересной, так как игры всегда пользуются высокой популярности, а в рамках университета, игра, которая может подточить ваши навыки вычислений, будет весьма полезна и увлекательна.

Перечень основных функций, предъявляемых к разрабатываемой сетевой игре:

1. *Игра должна происходить с помощью клиента и сервера. Генерация выражения и большая часть вычислений должны происходить на сервере.*
2. *Время ожидания на ответ игрока должно быть разным, в зависимости от сложности выражения.*
3. *Анализ ответов и присуждение победы игроку возлагается на сервер.*
4. *Отельные логические части программы должны быть выделены в отдельном потоке.*
5. *Сервер должен обладать рядом контролирующих команд.*
6. *Вывод клиентским приложением результатов игры.*

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА.

1.1 Постановка задачи.

Задачей данной курсовой работы является разработка сетевой игры «Соревновательная математика», которая рассчитана на несколько 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.

Основным дистрибутивом линукс, на котором будет работать игра выбран Ubuntu 14.04 на архитектуре i386, так как только этот дистрибутив установлен на учебных компьютерах в НовГУ.

Сетевая часть программы будет реализована с помощью библиотек сокетов Беркли на C++. Протокол передачи данных: TCP.

1.3 Выбор модели.

**Модель игры выглядит следующим образом**:

1. Сервер генерирует выражение. Отправляет игрокам время на раздумья и само выражение.
2. Игроку высвечивается выражение. Ему необходимо ответить за определённое время.
   1. Если игрок успевает вписать ответ, то этот ответ отсылается на сервер.
   2. Если не успел, то отсылается на сервер специальное число, не равное ответу на выражение.
3. Сервер анализирует ответы игроков. Здесь несколько случаев:
   1. Никто не ответил правильно. Никому не присудить победу.
   2. Имеется один правильный ответ. Присудить игроку победу.
   3. Имеется несколько правильных ответов. Проанализировать время ответа игроков, и присудить победу самому быстрому.
4. Если количество победных очков любого игрока достигло максимального, то закончить игру, иначе перейти к шагу 1.

1.4 Протокол взаимодействия клиента и сервера.

Протокол передачи данных – TCP. TCP выбран неспроста – в играх почти всегда используется этот протокол, так как приложение должно убедиться, что до второй стороны дошла информация. Иначе игра будет работать неверно, и из-за недошедшего пакета информации до одного пользователя, игра может приостановиться и у всех остальных.

1.5 Вывод по первой главе.

Таким образом, для реализации игры «Соревновательная математика» основной ОС выбрана Ubuntu, языком программирования процедурной части С++.

Были определены задачи, предъявляемые для программы.

Определена модель игры и взаимодействия клиента и сервера.

Был составлен протокол взаимодействия клиентов и сервера.

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА.

2.1 Реализация сервера.

*Подключение*

При запуске программы необходимо предоставить аргумент командной строки, который будет являться портом сервера. Сокет создаётся при помощи системного вызова socket(). Далее создаём структуру sockaddr\_in которая определяет настройки соединения и ставим поле sin\_family на AF\_INET, таким образом ставя подключение на TCP. Привязка соединения происходит с помощью функции bind(), listen() позволяет подключаться клиентам. Далее в main создаётся отдельный поток get\_clients, который в бесконечном цикле будет принимать входящих клиентов с помощью функции accept() и заносить их в специальный вектор структур Players, который будет использоваться для реализации потока игры.

*Команды сервера*

Сервер имеет набор команд:

* /start число\_очков – Запускает игру с заданным максимальным числом победных очков. Вместо число можно написать “inf”, чтобы игра длилась бексонечно.
* /disconnect айди\_клиента – Отключает клиента по предложенному ID. Ищет ID клиента среди вектора структур Players, в случае если находит, то отключает клиента по его сокету. Если не находит то выводит об этом сообщение.
* /loglevel число – Определяет уровень вывода сообщение. Уровень 0 – никуда не выводит, 1 – выводит в консоль, 2 – выводит в консоль и в файл “log”. По началу программы равен 1.

*Поток игры*

Как уже сообщалась, игра начнётся при вызове команд /start. В потоке активно используется вектор структур Players, в котором удобно хранить данные об игроках. Структура Players состоит из: идентификатора сокета игрока, ID игрока, игровые очки игрока, ответ игрока, время ответа игрока, и bool значение был ли ответ игрока верным.

Итак, при старте потока, сервер посылает всем игрокам bool флаг о том, что игра начинается. Объявляется переменная текущего максимального значения очков player\_max\_score игроков, которая равна по началу 0. Начинается цикл до тех пор, пока это число не станет равным максимальному возможному очку, которое мы вводили в команде /start. В начале этого цикла отправляется флаг о том, что игра продолжается – в ином случае, флаг об окончании игры будет отправлен, когда мы выйдем из этого цикла, но об этом позднее.

После отправки флага сервер вызывает функцию get\_expression файла Math.cpp, получая текстовое математическое выражение, его численный результат и сложность. Необходимо определить время, которое будет ждать сервер для ответа игрока. Оно определяется следующим образом – умножаем базовое время ожидания (5) на полученною сложность, приведённую к числу (enum’ы позволяют конвертацию в int, Лёгкая сложность – 1, Средняя – 2, Сложная – 3). Таким образом получаем 5, 10, или 15, в зависимости от сложности. Отсылаем время на раздумья и текстовое выражение всем игрокам.

Далее после приостановки программы на время ожидания ответа (с помощью функции sleep()), получаем ответ и время ответа от всех пользователей, запоминаем эту информацию, изменив поля игрока. Если ответ пользователя был верным, то ставим поле is\_right в структуре Players на true, запоминаем сокет этого игрока (будет полезно, если это будет единственный правильный ответ) и увеличиваем переменную правильных ответов.

После получения всех ответов нам необходимо проверить – единственный ли ответ был правильным или нет. Если правильный ответ не единственный, то нам необходимо проверить самого быстрого. Объявляем переменную минимального времени, сбрасываем победный сокет, после чего пробегаем по вектору всех игроков – если игрок ответил правильно, и его время ответа, меньше текущего минимального, то запомнить его сокет и изменить минимальное время на это время. Таким образом определим единственный сокет победного игрока.

Далее пробегаем по вектору всех пользователей и с помощью числовой переменной отправляем статус его ответа. Если ответ был неверным – ставим статус на 0. Если ответ был верным, но проиграл по времени – статус 1. Если игрок победил – то статус 2, и тут же увеличиваем его очки, и переменную player\_max\_score, в случае если эти очки максимальны. Отсылаем всем игрокам их статус.

Так же между раундами сбрасываем ответ пользователя, время ответа и bool переменную.

Далее если очки не достигли максимальных, то начинается новый раунд с флагом начатия раунда, иначе выходим из цикла раундов, отправляем флаг конца игры, и отправляем результаты игры – каждому пользователю присылаем информацию о каждом пользователе, состоящей из очков игрока и его ID. Поток заканчивает своё выполнение.

*Генерация математического выражения.*

Функция get\_expression() выполняет следующие действия:

1. Получить случайный знак с помощью функции rand() из enum Operand.
2. В соответствии с полученным знаком выполнить определить последовательность действий.
3. Например, если получили плюс, то:
   1. Сгенерировать левое число в пределах до 1000, правое до 100.
   2. Посчитать числовой результат выражения.
   3. Составить строку выражения в виде 100 + 56.
   4. Определить количество цифр двух полученных чисел.
   5. В зависимости от числа цифр определить сложность выражения.

Функция возвращает string (текстовое выражение), так как аргументы численного результата и сложности передаются по ссылке, то они изменятся и там, где функция была вызвана.

Каждый знак определяет свои пределы генерируемых чисел, и своя сложность, в зависимости от полученных цифр.

2.2 Реализация клиента.

*Подключение*

Клиент в качестве аргумента командной строки принимает аргумент вида ip:port, разделяет эту строку на строки ip и port, port приводит к численному значению с помощью функции stoi(), ip приводит к нужному с помощью вызова inet\_aton(). Далее получаем хостинг (сервер) с помощью функции gethostbyaddr(), предоставляя ip. Далее по полученному хостингу и сокету подключаемся к серверу с помощью функции connect().

*Поток игры*

Поток игры вызывается в функции int\_main с помощью вызова pthread\_create(). В этом потоке имеем цикл while (true), который отвечает за ожидание начала игры, и вложенный while (true) самой игры. Первый цикл ждёт флага от сервера и переходит во второй цикл. В нём мы ожидаем флаг о начале/конце раунда. Если получили флаг конца, то выходим из цикла. Иначе продолжаем цикл.

Далее от сервера мы получаем время на ответ и текстовое выражение, печатая его на экран. Затем запускаем поток ожидания ответа от пользователя, предварительно запомнив время начала ответа с помощью функции clock(), так как нам надо вычислить время, за которое пользователь дал ответ. Поток ожидания ответа сканирует ответ с консоли, предварительно вызвав функцию poll(), с переданным временем на ответ. Эта функция позволяет не дожидаться обязательного ответа от пользователя с помощью функции scanf() – если это время прошло, то ответ заполниться числом INPUT\_FAIL.

После выполнения потока ожидания ответа замеряем время конца ответа и находим общее время на ответ, вычитая из конечного начальное и разделив на константу CLOCKS\_PER\_SEC. Отсылаем ответ и время ответа серверу.

Далее сервер займётся анализом ответа, а клиент в свою очередь ожидает от него результата. Сравниваем пришедший результат – если она равен 0/1/2 выводим соответствующие сообщения.

Если игра на этом моменте заканчивается, то в моменте принятия флага продолжения игры, цикл закончиться, и тут мы должны принять информацию о результатах игры от сервера. Получаем количество игроков (чтобы правильно выполнить цикл приёма информации), и от 0 до этого числа принимаем информацию обо всех игроках – их ИД и очки – и выводим на экран.

2. 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. |
| gethostbyaddr () | Возвращает структуру типа hostent машине с именем name. В данном случае name является либо именем машины, либо адресом IPv4 в стандартной точечной нотации, либо адресом IPv6 в нотации с двоеточием (и, возможно, точкой) в качестве разделителя (см. описание адресов IPv6 в RFC 1884). Если name является адресом IPv4 или IPv6, то поиск не производится и gethostbyname() просто копирует name в поле h\_name, а его эквивалент для структуры struct in\_addr копируется в поле h\_addr\_list[0] возвращаемой структуры hostent. Если name не оканчивается точкой и установлена переменная окружения HOSTALIASES, то в файле, на который указывает HOSTALIASES, будет произведен поиск name (формат файла приведен в hostname(7)). Если name не оканчивается точкой, то будет произведен поиск в текущем домене и его родителях. |
| accept() | Извлекает первый запрос на соединение из очереди ожидающих соединений, создаёт новый подключенный сокет и выделяет для сокета новый файловый дескриптор, который и возвращается. В качестве параметров принимает дескриптор слушающего сокета, указатель на структуру адреса и размер структуры адреса. |
| inet\_aton() | Преобразовывает обычный вид IP-адреса cp (из номеров и точек) в двоичный код и сохраняет его в структуре, на которую указывает inp. inet\_aton возвращает ненулевое значение, если адрес правильный, и возвращает ноль, если он неверен. |

2.5 Результат работы.

В ходе выполнения данной курсовой работы удалось разработать сетевую игру «Соревновательная математика», которая соответствует поставленным им требованиям. Общение сервера и клиента происходит при помощи сокетов с типом подключения TCP. Сервер генерирует различные математические выражения, определяя их сложность, и выдаёт разное время на решение игрокам. Отдельные подзадачи выполняются в отдельных потоках. Общение происходит без ошибок. Сервер имеет определённый набор управляющих команд. В конце игры клиент выводит результаты в консоль.

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 g++. Сетевое взаимодействие было построено с помощью протокола TCP. Сервер может одновременно обслуживать несколько клиентов. Логика игры соответствует её прототипу. В целом созданная сетевая игра логически завершена.

**Листинги**

**Server.cpp**

|  |
| --- |
| #include <stdio.h> |
|  | #include <sys/types.h> |
|  | #include <sys/socket.h> |
|  | #include <netinet/in.h> |
|  | #include <stdlib.h> |
|  | #include <strings.h> |
|  | #include <unistd.h> |
|  | #include <string.h> |
|  | #include <map> |
|  | #include <threads.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <pthread.h> |
|  | #include <vector> |
|  | #include <ctime> |
|  | #include "Math.cpp" |
|  | #define SCORE\_INF -1 |
|  | const int THINK\_TIME = 5; |
|  |  |
|  | using namespace std; |
|  | int sockfd; // Сокет сервера |
|  | int log\_level = 1; |
|  |  |
|  | // Данные об игроке. Его сокет, ID, очки игры, ответ, время ответа, был ли ответ верным. |
|  | struct Player |
|  | { |
|  | int fd; |
|  | int id; |
|  | int score = 0; |
|  | int answer; |
|  | double time; |
|  | bool is\_right; |
|  | }; |
|  |  |
|  | vector<Player> players; |
|  |  |
|  | void\* game\_thread(void\*); |
|  | void\* server\_commands(void\*); |
|  | int refresh\_answers(); |
|  | int refresh\_round(); |
|  |  |
|  | void error(string msg) |
|  | { |
|  | perror(msg.c\_str()); |
|  | exit(1); |
|  | } |
|  |  |
|  | void log\_message(string message) |
|  | { |
|  | if (log\_level == 0) return; // если лог == 0 то выходим (ничего не выводим) |
|  | char cbuff[message.size()]; |
|  | strcpy(cbuff, message.c\_str()); |
|  | printf("%s\n", cbuff); |
|  | if (log\_level == 1) return; // если лог == 1 то выходим (вывели в консоль) |
|  | FILE\* file; // Тут получится что выведем и в консоль и в файл |
|  | file = fopen("log", "w+"); |
|  | if (file != NULL) |
|  | { |
|  | time\_t seconds = time(NULL); |
|  | tm\* timeinfo = localtime(&seconds); |
|  | fputs(asctime(timeinfo), file); |
|  | fputs(cbuff, file); |
|  | fflush(file); |
|  | } |
|  | } |
|  |  |
|  | // Начинает игру |
|  | void start\_game(int max\_score) |
|  | { |
|  | if (players.size() > 0) |
|  | { |
|  | pthread\_t game\_thr; |
|  | pthread\_create(&game\_thr, NULL, game\_thread, (void\*)&max\_score); // Запускаем поток игры с аргументом max\_score |
|  | } |
|  | } |
|  |  |
|  | // Поток игры |
|  | void\* game\_thread(void\* max\_scr) |
|  | { |
|  | srand(time(NULL)); // Сид рандома |
|  |  |
|  | int possible\_max\_score = \*(int\*)max\_scr; // приводим аргумент void\* к инт. Это максимально возможные очки |
|  | int players\_count = players.size(); // кол-во игроков |
|  | int player\_max\_score = 0; // текущее макс кол-во очков игроков |
|  |  |
|  | for (int i = 0; i < players\_count; i++) |
|  | { |
|  | bool game\_flag = true; |
|  | write(players[i].fd, &game\_flag, sizeof(bool)); // Отправляем флаг старта игры |
|  | } |
|  |  |
|  | log\_message("Игра началась! Макс очков " + to\_string(possible\_max\_score) + "\n\n"); |
|  |  |
|  | while (player\_max\_score != possible\_max\_score) // Цикл раунда пока макс. очки игрока не достигнут макс. возможных очков (например играем до 10 победных) |
|  | { |
|  | bool game\_end = false; |
|  | for (int i = 0; i < players.size(); i++) // Отправляем флаг того что игра продолжается |
|  | { |
|  | write(players[i].fd, &game\_end, sizeof(bool)); |
|  | } |
|  |  |
|  | int answer = 0; |
|  | Difficulty diff = Difficulty::Easy; |
|  | string expression = get\_expression(answer, diff); // Получаем выражение, ответ, сложность, передав последние по ссылке в функцию |
|  | char expbuf[expression.size()]; // Переводим string выражение в C буфер |
|  | strcpy(expbuf, expression.c\_str()); // Копирум string в C буфер |
|  |  |
|  | const int think\_time = THINK\_TIME \* (int)diff; // базовое ожидание умножаем на полученную сложность (1 легко, 2 средне, 3 сложно) |
|  | for (int i = 0; i < players\_count; i++) |
|  | { |
|  | write(players[i].fd, &think\_time, sizeof(int)); // отправляем всем игрокам время на решение |
|  | write(players[i].fd, expbuf, strlen(expbuf) \* sizeof(char)); // и само выражение |
|  | } |
|  |  |
|  | sleep(think\_time); // ждём время на решение |
|  |  |
|  | int winner\_fd; |
|  | int right\_answ\_cnt = 0; |
|  | for (int i = 0; i < players\_count; i++) // заполняем ответы игроков |
|  | { |
|  | int answ; |
|  | double time; |
|  | read(players[i].fd, &answ, sizeof(int)); // получаем ответ |
|  | read(players[i].fd, &time, sizeof(double)); // получаем время ответа |
|  |  |
|  | players[i].answer = answ; // ставим ответ игрока на полученный |
|  | players[i].time = time; // время игрока |
|  | if (answ == answer) // если ответ игрока равен правильному |
|  | { |
|  | players[i].is\_right = true; // ставим ему статус правильности |
|  | winner\_fd = players[i].fd; // ставим fd победного игрока как этого (если он будет единственным победным) |
|  | right\_answ\_cnt++; // счётчик правильных ответов увеличиваем |
|  | } |
|  | log\_message("Игрок " + to\_string(players[i].id) + " ответ: " + to\_string(answ) + " | время: " + to\_string(time)); |
|  | } |
|  |  |
|  | if (right\_answ\_cnt > 1) // если кол-во правильный ответов больше 1, то будет определять по минимальному времени ответа |
|  | { |
|  | winner\_fd = 0; // сбрасываем победного игрока |
|  | double best\_time = INT16\_MAX; // инициализируем переменную лучшего времени большим числом |
|  | for (int i = 0; i < players.size(); i++) // проверяем всех игроков |
|  | { |
|  | if (players[i].is\_right && players[i].time < best\_time) // если ответ игрока был правильным и его время меньше текущего лучшего |
|  | { |
|  | best\_time = players[i].time; // обновляем лучшее время |
|  | winner\_fd = players[i].fd; // обновляем fd победного игрока |
|  | } |
|  | } |
|  | } |
|  |  |
|  | for (int i = 0; i < players\_count; i++) // отсылаем результаты игрокам |
|  | { |
|  | int status = 0; // статус ответа игрока. 0 - неправильный, 1 - правильный но проиграл по времени, 2 - победил |
|  | if (players[i].is\_right && players[i].fd == winner\_fd) // если игрок был прав и его fd == победному fd |
|  | { |
|  | players[i].score++; // увеличиваем его очки |
|  | if (players[i].score > player\_max\_score) player\_max\_score = players[i].score; // обновляем максимальное значение очков. если достигнет possible\_max\_score, то выйдет из цикла игры |
|  | status = 2; // ставим статус на 2 (победный) |
|  | } |
|  | else if (players[i].is\_right && right\_answ\_cnt > 1) // если не прошло предыдущее условие на победность, но он всё равно дал правильный ответ |
|  | { |
|  | status = 1; // ставим статус на 1 |
|  | } // Если игрок был не прав то статус так и остаётся в 0 |
|  |  |
|  | write(players[i].fd, &status, sizeof(int)); // отправляем игроку его статус |
|  | } |
|  |  |
|  | for (int i = 0; i < players\_count; i++) // Выводим сообщение |
|  | { |
|  | log\_message("Игрок " + to\_string(players[i].id) + " очки: " + to\_string(players[i].score) + "\n"); |
|  | } |
|  |  |
|  | log\_message("Текущее максимальное очко: " + to\_string(player\_max\_score) + "\n"); |
|  | refresh\_answers(); // обновляем данные ответов игроков |
|  | } |
|  |  |
|  | bool game\_end = true; // Отправляем флаг конца игры |
|  | for (int i = 0; i < players.size(); i++) // Цикл для всех пользователей |
|  | { |
|  | write(players[i].fd, &game\_end, sizeof(bool)); // отсылаем флаг |
|  | write(players[i].fd, &players\_count, sizeof(int)); // отправляем количество игроков чтобы клиент вывел информацию о каждом |
|  | for (int j = 0; j < players.size(); j++) // Цикл в цикле. Отправим всем пользователям инфу а каждом пользователе (его id и очки) |
|  | { |
|  | int id = players[j].id; |
|  | int score = players[j].score; |
|  | write(players[i].fd, &id, sizeof(int)); // отправляем id |
|  | write(players[i].fd, &score, sizeof(int)); // отправляем очки |
|  | } |
|  | } |
|  | refresh\_round(); // обновляем очки |
|  | } |
|  |  |
|  | void\* server\_commands(void\*) |
|  | { |
|  | while (true) |
|  | { |
|  | char command[16]; |
|  | scanf("%s", command); |
|  | if (strcmp(command, "/start") == 0) // начинает игру |
|  | { |
|  | memset(command, 0, 16); // сбрасываем буфер |
|  | scanf("%s", command); |
|  |  |
|  | int max\_score; |
|  | if (strcmp(command, "inf") == 0) // бесконечно кол-во очков |
|  | { |
|  | max\_score = SCORE\_INF; |
|  | } |
|  | else |
|  | { |
|  | max\_score = atoi(command); // в численный вид |
|  | if (max\_score <= 0) // если написал 0 или меньше |
|  | { |
|  | printf("Количество победных очков должно быть больше 0!\n"); |
|  | continue; |
|  | } |
|  | } |
|  |  |
|  | start\_game(max\_score); // начинаем игру |
|  | } |
|  | else if (strcmp(command, "/disconnect") == 0) // отключить пользователя |
|  | { |
|  | memset(command, 0, 16); // сбрасываем буфер |
|  | scanf("%s", command); |
|  |  |
|  | int client\_id; |
|  | client\_id = atoi(command); // id пользователя в число |
|  | int client\_fd = -1; |
|  | int index; |
|  |  |
|  | for (int i = 0; i < players.size(); i++) // ищем среди пользователей пользователя с нужным id |
|  | { |
|  | if (players[i].id == client\_id) |
|  | { |
|  | index = i; |
|  | client\_fd = players[i].fd; // запоминаем его сокет если нашли |
|  | break; |
|  | } |
|  | } |
|  | if (client\_fd != -1) // Если нашли |
|  | { |
|  | close(client\_fd); // Закрываем сокет |
|  | players.erase(players.begin() + index); // удаляем игрока |
|  | printf("Разорвано соединение с клентом %i\n", client\_id); |
|  | } |
|  | else // Если не нашли |
|  | { |
|  | printf("Такого id клиента не существует!\n"); |
|  | } |
|  | } |
|  | else if (strcmp(command, "/loglevel") == 0) // Поставить уровень вывода |
|  | { |
|  | memset(command, 0, 16); // сброс буфера |
|  | scanf("%s", command); |
|  | log\_level = atoi(command); // перевод в число |
|  |  |
|  | printf("Уровень вывода успешно поставлен на %i\n", log\_level); |
|  | } |
|  | } |
|  | } |
|  |  |
|  | // Поток получения клиентов |
|  | void\* get\_clients(void\*) |
|  | { |
|  | socklen\_t clilen; |
|  | sockaddr\_in cli\_addr; // адрес клиента |
|  | clilen = sizeof(cli\_addr); |
|  |  |
|  | int client\_id = 1; |
|  | while (true) |
|  | { |
|  | int client\_fd = accept(sockfd, (sockaddr\*)&cli\_addr, &clilen); // Принимаем клиентов |
|  | if (client\_fd < 0) error("Ошибка при подключении клиента!\n"); // Если ошибка |
|  | printf("Подключился игрок с ID %i\n", client\_id); |
|  |  |
|  | Player new\_player{client\_fd, client\_id, 0}; // создаём нового игрока |
|  | players.push\_back(new\_player); // заносим в вектор players |
|  | write(client\_fd, &client\_id, sizeof(int)); // отправляем игроку его id |
|  |  |
|  | client\_id++; // увеличиваем ID |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char\* argv[]) |
|  | { |
|  | setlocale(LC\_ALL, "Russian"); |
|  | int port; |
|  | if (argc >= 2) // если аргументы есть |
|  | { |
|  | port = atoi(argv[1]); // строку порта в число |
|  | } |
|  | else |
|  | { |
|  | error("Укажите порт!\n"); |
|  | } |
|  |  |
|  | /\* Создаём сокет \*/ |
|  | sockfd = socket(AF\_INET, SOCK\_STREAM, 0); |
|  | if (sockfd < 0) error("Ошибка создания сокета!\n"); |
|  | /\* \*/ |
|  |  |
|  | /\* Создаём соединение \*/ |
|  | sockaddr\_in server\_adr; |
|  | bzero((char\*)&server\_adr, sizeof(server\_adr)); |
|  | server\_adr.sin\_addr.s\_addr = INADDR\_ANY; |
|  | server\_adr.sin\_family = AF\_INET; |
|  | server\_adr.sin\_port = htons(port); |
|  | if (bind(sockfd, (sockaddr\*)&server\_adr, sizeof(server\_adr)) < 0) error("Ошибка создания соединения\n!"); |
|  | /\* \*/ |
|  |  |
|  | if (listen(sockfd, 5) < 0) error("Ошибка на listen()!\n"); // даём доступ на подключение клиентов |
|  |  |
|  | printf("Сервер запущен на %i порте\n", port); |
|  |  |
|  | pthread\_t thr1, thr2; |
|  | pthread\_create(&thr1, NULL, get\_clients, NULL); // поток получения клиентов |
|  | pthread\_create(&thr2, NULL, server\_commands, NULL); // поток команд |
|  | pthread\_join(thr2, NULL); // присоединение к потоку |
|  |  |
|  | } |
|  |  |
|  | int refresh\_answers() |
|  | { |
|  | for (int i = 0; i < players.size(); i++) |
|  | { |
|  | players[i].answer = 0; |
|  | players[i].time = 0; |
|  | players[i].is\_right = false; |
|  | } |
|  | } |
|  |  |
|  | int refresh\_round() |
|  | { |
|  | refresh\_answers(); |
|  | for (int i = 0; i < players.size(); i++) |
|  | { |
|  | players[i].score = 0; |
|  | } |
|  | } |
|  |  |
|  | bool max\_score\_check(int max\_score) |
|  | { |
|  | for (int i = 0; i < players.size(); i++) |
|  | { |
|  | if (players[i].score >= max\_score) return true; |
|  | } |
|  | return false; |
|  | } |

Client.cpp

|  |
| --- |
| #include <stdlib.h> |
|  | #include <stdio.h> |
|  | #include <sys/types.h> |
|  | #include <sys/socket.h> |
|  | #include <netinet/in.h> |
|  | #include <arpa/inet.h> |
|  | #include <netdb.h> |
|  | #include <strings.h> |
|  | #include <unistd.h> |
|  | #include <cstring> |
|  | #include <map> |
|  | #include <string> |
|  | #include <poll.h> |
|  | #include <vector> |
|  |  |
|  | using namespace std; |
|  | #define INPUT\_FAIL INT16\_MIN |
|  |  |
|  | int sockfd; |
|  |  |
|  | void error(string msg) |
|  | { |
|  | perror(msg.c\_str()); |
|  | exit(1); |
|  | } |
|  |  |
|  | void\* wait\_for\_answer(void\* time) |
|  | { |
|  | int answer\_time = \*(int\*)time; |
|  | int answer; |
|  | pollfd mypoll = { STDIN\_FILENO, POLLIN|POLLPRI }; // переменная для ожидания записи в консоль |
|  |  |
|  | if (poll(&mypoll, 1, answer\_time \* 1000)) // если за время answer\_time \* 1000 (5 \* 1000 = 5 сек) получим запись в консоль |
|  | { |
|  | scanf("%i", &answer); // то прочитываем с консоли |
|  | } |
|  | else // иначе ставим на INPUT\_FAIL |
|  | { |
|  | answer = INPUT\_FAIL; |
|  | } |
|  |  |
|  | int\* result = new int; // выделяем память чтобы вернуть значение |
|  | \*result = answer; |
|  | return (void\*)result; // возвращаем результат с помощью выделенной памяти |
|  | } |
|  |  |
|  | // Поток игры |
|  | void\* game\_thread(void\*) |
|  | { |
|  | while (true) // Бесконечный цикл ождиания начала игры |
|  | { |
|  | bool game\_flag; |
|  | read(sockfd, &game\_flag, sizeof(bool)); // Принимаем флаг начала игры |
|  | if (game\_flag != true) error("Ошибка начала игры.\n"); // Сообщение если что-то не так |
|  |  |
|  | while (true) // Цикл игры |
|  | { |
|  | bool game\_end; |
|  | read(sockfd, &game\_end, sizeof(bool)); // Принимаем флаг продолжается игра или кончилась |
|  | if (game\_end) break; // если кончилась то выходим из цикла игры |
|  |  |
|  | int think\_time; |
|  | char expression[20]; // текстовое выражение |
|  | read(sockfd, &think\_time, sizeof(int)); // прочитываем время на решение с сервера |
|  | read(sockfd, expression, sizeof(char) \* 20); // читаем выражение с сервера |
|  |  |
|  | printf("\nНа размышление даётся %i секунд.\n", think\_time); |
|  | printf("%s = ", expression); // пишем выражение |
|  | fflush(stdout); // чтобы курсор не переносился после принтф |
|  |  |
|  | memset(expression, 0, sizeof(char) \* 20); // сбрасываем выражение чтобы не выводил старое в консоль |
|  |  |
|  | int\* answer; |
|  | pthread\_t thr; |
|  | clock\_t start = clock(); // получаем время начала ввода ответа |
|  | pthread\_create(&thr, NULL, wait\_for\_answer, (void\*)&think\_time); // поток для получения ответа с консоли |
|  | pthread\_join(thr, (void\*\*)&answer); // присоединяемся к нему |
|  | clock\_t end = clock(); // время конца ответа |
|  | printf("\n"); |
|  | //int actual\_answer = \*answer; |
|  |  |
|  | double time = (end - start) / CLOCKS\_PER\_SEC; // вычисляем время ответа |
|  | write(sockfd, answer, sizeof(int)); // отсылаем ответ на сервер |
|  | write(sockfd, &time, sizeof(double)); // отсылаем время ответа |
|  |  |
|  | delete answer; |
|  |  |
|  | int result; |
|  | read(sockfd, &result, sizeof(int)); // читаем результат ответа с сервера |
|  | if (result == 0) //если 0 |
|  | { |
|  | printf("Вы ответили неправильно!\n"); |
|  | } |
|  | else if (result == 1) // если 1 |
|  | { |
|  | printf("Вы ответили правильно, но кто-то ответил быстрее!\n"); |
|  | } |
|  | else if (result == 2) //если 2 |
|  | { |
|  | printf("Вы победили! +1 победное очко\n"); |
|  | } |
|  | } |
|  |  |
|  | // Сюда мы попадём если выйдем с цикла игры. Выводим результаты игры. |
|  | printf("Раунд закончился.\n\n Результаты:\n"); |
|  | int player\_cnt; |
|  | read(sockfd, &player\_cnt, sizeof(int)); // получаем кол-во игроков |
|  | for (int i = 0; i < player\_cnt; i++) // цикл по кол-ву игрохов |
|  | { |
|  | int id; |
|  | int score; |
|  | read(sockfd, &id, sizeof(int)); // получаем id игрока |
|  | read(sockfd, &score, sizeof(int)); // и очки игрока |
|  | printf("%i Игрок: %i победных очков.\n", id, score); // выводим результат |
|  | } |
|  | } |
|  | } |
|  |  |
|  | void\* commands\_thread(void\*) |
|  | { |
|  | while (true) |
|  | { |
|  |  |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | setlocale(LC\_ALL, "Russian"); |
|  |  |
|  | if (argc < 2) |
|  | { |
|  | error("Предоставьте адрес сервера!\n"); |
|  | } |
|  |  |
|  | char\* adress = argv[1]; |
|  | string ip\_str; |
|  | string port\_str; |
|  | int i = 0; |
|  | for (i; i < strlen(adress); i++) |
|  | { |
|  | if (adress[i] == ':') break; |
|  | ip\_str += adress[i]; |
|  | } |
|  | i++; |
|  | for (i; i < strlen(adress); i++) |
|  | { |
|  | port\_str += adress[i]; |
|  | } |
|  |  |
|  | /\* Создаём сокет \*/ |
|  | sockfd = socket(AF\_INET, SOCK\_STREAM, 0); |
|  | if (sockfd < 0) error("Error opening socket!"); |
|  | /\* \*/ |
|  |  |
|  | /\* Получаем сервер (хост) по ip \*/ |
|  | in\_addr ip; |
|  | hostent \*server; |
|  | if (!inet\_aton(ip\_str.c\_str(), &ip)) error("Проверьте правильность адреса!\n"); |
|  | if ((server = gethostbyaddr((const void\*)&ip, sizeof(ip), AF\_INET)) == NULL) error("Не существует сервера по адресу " + ip\_str + "\n"); |
|  | /\* \*/ |
|  |  |
|  | /\* Подключаемся к серверу по хосту \*/ |
|  | int port = stoi(port\_str); // Приводим строку порта к численному виду |
|  | sockaddr\_in serv\_addr; |
|  | bzero((char\*)&serv\_addr, sizeof(serv\_addr)); |
|  | bcopy((char\*)server->h\_addr\_list[0], (char\*)&serv\_addr.sin\_addr.s\_addr, server->h\_length); |
|  | serv\_addr.sin\_family = AF\_INET; |
|  | serv\_addr.sin\_port = htons(port); |
|  | if (connect(sockfd, (sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) error("Ошибка подключения к серверу\n"); |
|  | /\* \*/ |
|  |  |
|  | printf("Успешное подключение! Ожидание начала игры.\n"); |
|  |  |
|  | int id = 0; |
|  | read(sockfd, &id, sizeof(int)); // получаем id от сервера |
|  | printf("Ваше ID: %i", id); |
|  |  |
|  | pthread\_t thr1, thr2; |
|  | pthread\_create(&thr1, NULL, commands\_thread, NULL); |
|  | pthread\_create(&thr2, NULL, game\_thread, NULL); |
|  | pthread\_join(thr1, NULL); |
|  | } |

Math.cpp

|  |
| --- |
| #include <string> |
|  |  |
|  | enum class Operands |
|  | { |
|  | Plus = 0, Minus = 1, Multiply = 2, Divide = 3 |
|  | }; |
|  |  |
|  | enum class Difficulty |
|  | { |
|  | Easy = 1, Medium = 2, Hard = 3 |
|  | }; |
|  |  |
|  | int get\_digits(int number) |
|  | { |
|  | int i = 0; |
|  | for (i; number > 0; i++) number /= 10; |
|  | return i; |
|  | } |
|  |  |
|  | // Получить математическое выражение, его результат, его сложность |
|  | std::string get\_expression(int& result, Difficulty& diff) |
|  | { |
|  | Operands operand = Operands(rand() % 4); // Рандомим оператор от 0 до 3 включая (знак) |
|  | int operand1, operand2; |
|  | int digits = 0; // Количество цифр |
|  | std::string res; // Текстовый результат |
|  | switch (operand) |
|  | { |
|  | case Operands::Plus: // Получили плюс |
|  | operand1 = rand() % 1000; // первый операнд от 0 до 1000 (левое число) |
|  | operand2 = rand() % 100; // второй от 0 до 100 (правое число) |
|  | result = operand1 + operand2; // получаем числовой результат |
|  | res = std::to\_string(operand1) + " + " + std::to\_string(operand2); // создаём строку из выражения to\_string переводит число в строку |
|  | digits += get\_digits(operand1); // разбиваем число, разделяя его на 10, пока больше ноля. Таким образом находим кол-во цифр |
|  | digits += get\_digits(operand2); // то же самое |
|  | if (digits = 7) diff = Difficulty::Medium; // если кол-во цифр 7, то сложность средняя |
|  | else if (digits == 6) diff = Difficulty::Medium; // 6 - средняя |
|  | else diff = Difficulty::Easy; // остальное лёшкое |
|  |  |
|  | return res; |
|  |  |
|  | case Operands::Minus: // Получили минус. То же самое как с плюсом, только в строке минус и результат вычитания |
|  | operand1 = rand() % 1000; |
|  | operand2 = rand() % 1000; |
|  | result = operand1 - operand2; |
|  | res = std::to\_string(operand1) + " - " + std::to\_string(operand2); |
|  | digits += get\_digits(operand1); // разбиваем число, разделяя его на 10, пока больше ноля. Таким образом находим кол-во цифр |
|  | digits += get\_digits(operand2); // то же самое |
|  | if (digits >= 7) diff = Difficulty::Hard; // высокая |
|  | else if (digits >= 5) diff = Difficulty::Medium; |
|  | else diff = Difficulty::Easy; |
|  |  |
|  | return res; |
|  |  |
|  | case Operands::Multiply: // Умножение |
|  | operand1 = 2 + rand() % 10; // от 2 до 10 |
|  | operand2 = 2 + rand() % 10; // от 2 до 10 |
|  | result = operand1 \* operand2; |
|  | res = std::to\_string(operand1) + " \* " + std::to\_string(operand2); |
|  | digits += get\_digits(operand1); // разбиваем число, разделяя его на 10, пока больше ноля. Таким образом находим кол-во цифр |
|  | digits += get\_digits(operand2); // то же самое |
|  | if (digits == 4) diff = Difficulty::Medium; // если 4 то норм |
|  | else Difficulty::Easy; |
|  |  |
|  | return res; |
|  |  |
|  | case Operands::Divide: // Деление |
|  | operand1 = 2 + rand() % 100; |
|  | operand2 = 2 + rand() % 10; |
|  | while (operand2 > operand1) operand2 = 2 + rand() % 10; // Если операнд 2 больше операнда 1 (например 3 / 7), то перерандомим, т.к. будет получаться 0 |
|  | result = operand1 / operand2; |
|  | res = std::to\_string(operand1) + " / " + std::to\_string(operand2); |
|  | digits += get\_digits(operand1); // разбиваем число, разделяя его на 10, пока больше ноля. Таким образом находим кол-во цифр |
|  | digits += get\_digits(operand2); // то же самое |
|  | if (digits > 4) diff = Difficulty::Hard; |
|  | else if (digits == 4) diff = Difficulty::Medium; |
|  | else diff = Difficulty::Easy; |
|  |  |
|  | return res; |
|  |  |
|  | default: |
|  | break; |
|  | } |
|  | } |