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 «Аукцион»**

Курсовая работа

по дисциплине:

«Операционные системы»

Разработал:

Студент группы 9091

Чалый С.М \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2021г.

Проверил:

Ананьев В.В.\_\_\_\_\_\_

«\_\_».\_\_\_\_\_.2021г.

**Великий Новгород**

**2021**

ВВЕДЕНИЕ

Целью данной работы является создание клона сетевой игры “Аукциона”.

Особенность проектируемой игры заключается в том, что она должна работать под операционной системой Linux. Реализована игра на языке C++ и использованием фреймворка Qt.

Концепция игры такова. Всем играющим игрокам начисляются валюты для ставок на аукцион в будущем. Так же игроки могут получать деньги, нажимая на соответствующие кнопки. При старте аукциона перед игроками всплывает соответствующая форма. В ней игроки должны вписать ставку, которую они хотят поставить на аукцион, а также выбрать валюту для оплаты. Так же игроки могут спасовать, если они хотят пропустить текущий раунд. Одерживает победу и получает товар в раунде тот, кто поставил больше всех.

Перечень основных функций к разрабатываемой сетевой игре:

1. Симуляция вышеописанной игры.
2. Понятный графический интерфейс.
3. Обработка разных решений игроков.
4. Связь клиента и сервером.
5. Количество игроков от 1 до 4.

ГЛАВА 1. РАЗРАБОТКА ПРОЕКТА.

1.1 Постановка задачи.

Задачей данной курсовой работы является разработка сетевой игры «Аукциона», которая рассчитана на одного или 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 20.04 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.

Для реализации графического интерфейса было решено использовать кроссплатформенный инструментарий разработки Qt и входящий в его состав декларативный язык программирования QML, предназначенный для дизайна приложений и делающий основной упор на пользовательский интерфейс.

1.3 Выбор модели.

В сетевой игре, в которой может принять участие несколько игроков, необходима определенная модель, которая обеспечит их взаимодействие в игре. Для этого была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер. Сервер должен принимать и обрабатывать клиентов. Во время игры сервер получает информацию от игроков об их решении, их ставке, их валюте, выполняет необходимые вычисления и отправляет результаты всем игрокам. Отдельные подзадачи в клиенте и сервере обрабатываются в отдельных потоках.

1.4 Протокол взаимодействия клиента и сервера.

В качестве протокола взаимодействия выбран протокол TCP. Я решил использовать протокол TCP, так как он более приемлем для сетевых игр – мы должны убедиться, в том, что все пакеты достигли своего адресата. Иначе будут искажения информации и deadlock’и. Для отправки и получения информации используются функции send и recv. Информация отправляется разная – текстовый буфер, численные переменные.

1.5 Вывод по первой главе.

Таким образом, для реализации игры «Аукцион» основной ОС выбрана Ubuntu, языком программирования процедурной части С++, языком программирования графического интерфейса QML.

Была выбрана модель взаимодействия компьютеров и программ в сети клиент-сервер.

Был составлен протокол взаимодействия клиентов и сервера.

ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА.

**2.1 Реализация сервера.**

Сервер создаёт сокет, получая его идентификатор, используя функцию socket. Затем создаётся переменная типа sockaddr\_in с типом подключения AF\_INET (TCP) и портом из командной строки. С помощью функции bind привязываем адрес. Функция listen позволяет подключаться клиентам. Максимум 5 клиентов для подключения. Далее в бесконечном цикле вызываем функцию accept, которая принимает подключение клиента. Если число клиентом достигло MIN\_USERS, то с помощью функции pthread\_create, создадим поток auction\_thread, который обрабатывает поток игры. В этом потоке имеем бесконечный цикл, в котором поочерёдно выполняются следующие действия:

* + - 1. Генерируем время для ответа на аукцион. Ждём Время задержки то начала нового аукциона.
      2. По завершении ожидания отсылаем игрокам сообщение о старте аукциона и время для ответа.
      3. Ждём время для ответа.
      4. Далее в цикле for получаем результат игрока в виде текстового буфера.
      5. Если сообщение не равно “pas”, то разбиваем строку на ставку и валюту. Если валюта равна Usd или Eur, то переводим в рубли. Полученное число сравниваем с текущей лучшей ценой, для определения победителя. Если цена игрока лучше лучшей, то запоминаем его ID.
      6. Отсылаем всем игрокам результат текстовым буфером в виде ‘T’ / ‘F’ + сообщения для вывода.
      7. Переход к шагу 1.

**2.2 Реализация клиента.**

Рассмотрим файл Client.cpp. Сначала идёт создание окна Qt, суть которого рассмотрим позже. Клиент так же создаёт сокет, получает хостинг по ip (стандартному или из аргументов командной строки) и подключается по полученному хостингу, с помощью функции connect. Далее создаём поток прибавки валют money\_increase и auction\_handle. Money\_increase просто увеличивает валюты в бесконечном цикле каждую секунду. Поток auction\_handle имеет бесконечный цикл, в котором выполняются следующие действия:

1. Ждём сообщения о старте аукциона от сервера.
2. Получаем от сервера время на решение.
3. Далее запустим поток аукциона в Qt окне. Оно дождётся записи в поле ставки и вернёт результат.
4. Отправляем этот результат серверу.
5. Получаем результат выигрыша/проигрыша от сервера.
6. Если мы выиграли, то уменьшаем валюту, используемую для ставки.
7. Обновляем поля показывающие валюты.

**2.3 Реализация графического приложения**

Главные задачи от графического интерфейса: его наполненность, но простота; обновление данных; поля для записи информации серверу. Графическая часть интерфейса выполнена с помощью Qt creator. Суть его в том, что из пула объектов выбираются нужные и перетаскиваются в окно на необходимое местоположение. Так же элементам можно задать свойства, такие как размер, цвет, шрифт, css свойства и другое.

Изображение выглядит как текст, компьютер, ноутбук, внутренний

Автоматически созданное описание

Логическая часть интерфейса реализована в файлах mainwindow.h и mainwindow.cpp. Mainwindow.h это заголовочный файл окна – он содержит объявления всех функций для работы с окном. Категория slots нужна для вызова этих функций кнопками в окне. Mainwindow.cpp содержит конструктор (создание) главного окна. В нём происходит поиск всех необходимых элементов окна по их имени, заданном в Qt creator. Так же с помощью функции connect приписывается функционал кнопкам. Например, connect(getRubleBtn, SIGNAL(released()), this, SLOT(get\_ruble())) говорит о том, что кнопка getRubleButton при нажатии (released()) будет вызывать функцию get\_ruble(). В конце запускаем поток обновления полей денег.

Все “виджеты” Qt имеют встроенные функции-методы, с помощью которых мы можем менять их значение и поведение. Рассмотрим поток обновления полей денег. Мы используем числа rubles, dollars, euros, значение которых контролирует Client.cpp, они переводятся в строку, и выступают в качестве аргумента функции setText соответствующих полей денег в окне. В потоке текст меняется каждые 5 секунд.

Так же в этом файле есть поток auction\_thread, запускаемый Client.cpp во время ожидания ответа от пользователя. Этот поток проверяет, записанное пользователем значение ставки, выбранную валюту, анализирует на ошибки, и отправляет результат Client.cpp, который в свою очередь отправит его серверу.

Присутствуют функции конвертации одной валюты в другую, которые проверяют значение в нужном поле, и если ошибок нет, то вызывает функцию конвертации из файла Convert.cpp.

Кнопки увеличения денег прибавляют значение к числу rubles/dollars/euros, в зависимости от нажатой кнопки и обновляют текстовый показатель валюты.

Выключение и включение отдельных компонентов окна происходит с помощью функции виджета setVisible (true/false).

2.4 Системные вызовы.

Для данного проекта использовались следующие системные вызовы:

|  |  |
| --- | --- |
| Системный вызов | Описание |
| 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 «Аукциона», соответствующую поставленным к ней требованиям. Сервер и клиент успешно обмениваются информацией без перебоев. Получилось обработать возможные ошибки пользователя и обрабатывать различные ответы. Удалось разделить логику клиента и графического интерфейса. Удалось разделить проект на разные логические файлы. Графический интерфейс приложения удалось реализовать на декларативном языке программирования QML. Сервер и клиент удалось реализовать в разных потоках исполнения.

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. В целом созданная сетевая игра логически завершена.

**Листинги**

Файл 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 <threads.h> |
|  | #include <sys/stat.h> |
|  | #include <fcntl.h> |
|  | #include <pthread.h> |
|  | #include <vector> |
|  | #include <string> |
|  | #include "time.h" |
|  | #include "Convert.cpp" |
|  |  |
|  | #define MIN\_USERS 1 |
|  | using namespace std; |
|  | std::vector<int> connected\_clients; |
|  |  |
|  | int create\_socket\_serv() |
|  | { |
|  | int sockfd = socket(AF\_INET, SOCK\_STREAM, 0); |
|  | if (sockfd < 0) error("Error opening socket!"); |
|  | return sockfd; |
|  | } |
|  |  |
|  | sockaddr\_in create\_connection\_server(in\_port\_t port, int sockfd) |
|  | { |
|  | sockaddr\_in serv\_addr; |
|  | bzero((char\*)&serv\_addr, sizeof(serv\_addr)); |
|  | serv\_addr.sin\_addr.s\_addr = INADDR\_ANY; |
|  | serv\_addr.sin\_family = AF\_INET; // TCP |
|  | serv\_addr.sin\_port = htons(port); |
|  | if (bind(sockfd, (sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) error ("Error on binding!"); |
|  | } |
|  |  |
|  | void start\_listening(int sockfd) |
|  | { |
|  | if (listen(sockfd, 5) < 0) error("Error on listening start"); |
|  | } |
|  |  |
|  | // Поток игры |
|  | void\* auction\_thread(void\*) |
|  | { |
|  | while (true) |
|  | { |
|  | int auctionDelay = 10; // время до старта аукциона |
|  | int clientDecisionTime = rand() % 30 + 5; // время на решение пользователя |
|  | //int productPrice = rand() % 1000 + 500; // всегда в рублях |
|  |  |
|  | printf("Auction will start in %i seconds\n", auctionDelay); |
|  | sleep(auctionDelay); // Ждём время до старта аукциона |
|  | for (int clientId : connected\_clients) // оповещение всех подключённых пользователей |
|  | { |
|  | send(clientId, "Auction started", 127, 0); // оповещение об аукционе |
|  | send(clientId, to\_string(clientDecisionTime).c\_str(), 127, 0); // время на решение |
|  | } |
|  | printf("Auction started: %i for users decision", clientDecisionTime); |
|  | sleep(clientDecisionTime + 1); // ждём окончания аукциона |
|  |  |
|  | int bestPrice = -1; |
|  | int bestPriceId = -1; |
|  | for (int clientId : connected\_clients) |
|  | { |
|  | char buf[128]; |
|  | recv(clientId, &buf, 127, 0); // получаем решение пользователя |
|  | printf("%i client decision: %s", clientId, buf); |
|  | if (strcmp(buf, "pas") != 0) // если не пас |
|  | { |
|  | int price = atoi(buf + 1); // получаем цену |
|  | if (buf[0] == 'R') // валюта рубль |
|  | { |
|  |  |
|  | } |
|  | else if (buf[0] == 'U') // доллар |
|  | { |
|  | price = UsdToRub(price); // конвертируем в рубли (чтобы сравнить лучшую цену) |
|  | } |
|  | else if (buf[0] == 'E') // евро |
|  | { |
|  | price = EurToRub(price); // конвертируем в рубли |
|  | } |
|  |  |
|  | if (price > bestPrice) // если цена пользователя больше предыдущей лучшей |
|  | { |
|  | bestPrice = price; // лучшая цена равна этой |
|  | bestPriceId = clientId; // ID клиента лучшей цены |
|  | } |
|  | } |
|  | } |
|  | for (int clientId : connected\_clients) // Отсылаем результаты всем клиентам |
|  | { |
|  | char message[128]; |
|  | if (clientId != bestPriceId) sprintf(message, "FYou didn't bought goods :("); // не купил |
|  | else sprintf(message, "TYou bought goods for %i rubles!", bestPrice); // купил |
|  | printf("%i client will get the information: %s", clientId, message); // вывод в консоль |
|  | send(clientId, message, 127, 0); // отправляем результат |
|  | } |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char\* argv[]) |
|  | { |
|  | srand(time(NULL)); // ставим сид рандома |
|  | in\_port\_t portno = 2400; // стандартный порт |
|  | int sockfd, newclient; |
|  | socklen\_t clilen; |
|  | sockaddr\_in cli\_addr; |
|  | if (argc == 2) |
|  | { |
|  | portno = atoi(argv[1]); // если предоставлен порт из командной строки то переводим его в число и используем |
|  | } |
|  |  |
|  | sockfd = create\_socket\_serv(); // создаём сокет |
|  | create\_connection\_server(portno, sockfd); // создаём соединение |
|  | start\_listening(sockfd); // позволяем подключиться клиентам |
|  |  |
|  | clilen = sizeof(cli\_addr); |
|  |  |
|  | printf("Server started\n"); |
|  | printf("Waiting for %i users to connect\n", MIN\_USERS); // MIN\_USERS - минимальное кол-во пользователей для игры |
|  |  |
|  | int clientNum = 0; |
|  | pthread\_t auction\_thread\_id; |
|  | while (true) |
|  | { |
|  | newclient = accept(sockfd, (sockaddr\*)&cli\_addr, &clilen); // Принимаем клиента |
|  | if (newclient < 0) error("Error on accept\n"); // Если ошибка |
|  |  |
|  | connected\_clients.push\_back(newclient); // Заносим клиента в вектор connecter\_clients |
|  | clientNum++; // Номер клиента |
|  | if (MIN\_USERS - clientNum > 0) // Если достаточное число не подключилось |
|  | { |
|  | printf("Waiting for %i users to connect\n", MIN\_USERS - clientNum); |
|  | } |
|  | else // Если набралось,то начинаем игру |
|  | { |
|  | printf("Minimal %i users connected! Starting a game.\n", MIN\_USERS); |
|  | pthread\_create(&auction\_thread\_id, NULL, auction\_thread, NULL); // создаём поток игры |
|  | } |
|  | } |
|  | } |

Файл 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 <string> |
|  | #include "Convert.h" |
|  | #include "mainwindow.h" |
|  | #include "ui\_mainwindow.h" |
|  |  |
|  | #include <QApplication> |
|  | #include <QLocale> |
|  | #include <QTranslator> |
|  | #include <QLabel> |
|  |  |
|  | using namespace std; |
|  | #define STD\_PORT 2048 |
|  |  |
|  | double rubles = 0; |
|  | double dollars = 0; |
|  | double euros = 0; |
|  | int goods = 0; |
|  | int sockfd; |
|  |  |
|  | int create\_socket() |
|  | { |
|  | int sockfd = socket(AF\_INET, SOCK\_STREAM, 0); |
|  | if (sockfd < 0) error("Error opening socket!"); |
|  | return sockfd; |
|  | } |
|  |  |
|  | hostent\* get\_host\_by\_ip() |
|  | { |
|  | in\_addr ip; |
|  | hostent \*hp; |
|  | if (!inet\_aton(ipstr.c\_str(), &ip)) error("Can't parse IP address!"); |
|  | if ((hp = gethostbyaddr((const void\*)&ip, sizeof(ip), AF\_INET)) == NULL) error("No server associated with " + ipstr); |
|  | return hp; |
|  | } |
|  |  |
|  | sockaddr\_in server\_connect(in\_port\_t port, int sockfd, hostent\* server) |
|  | { |
|  | sockaddr\_in serv\_addr; |
|  | memset(&serv\_addr, 0, 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; // TCP |
|  | serv\_addr.sin\_port = htons(port); |
|  | //if (bind(sockfd, (sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) error ("Error on binding!"); |
|  | if (connect(sockfd, (sockaddr\*)&serv\_addr, sizeof(serv\_addr)) < 0) error("Error connecting"); |
|  | return serv\_addr; |
|  | } |
|  |  |
|  | // Поток увеличения денег игрока. Через секунду прибавляем каждой валюты. |
|  | void\* money\_increase(void\*) |
|  | { |
|  | const double rubDelta = 1; |
|  | const double dolDelta = 0.1; |
|  | const double euroDelta = 0.01; |
|  | while (true) |
|  | { |
|  | rubles += rubDelta; |
|  | dollars += dolDelta; |
|  | euros += euroDelta; |
|  | sleep(1); |
|  | } |
|  | } |
|  |  |
|  | // Поток игры |
|  | void\* auction\_handle(void\* arg) |
|  | { |
|  | MainWindow\* window = (MainWindow\*)arg; // Переводим аргумент типа void\* в QT окно MainWindow |
|  | int fd = sockfd; // сокет сервера |
|  | while (true) // Бесконечный цикл |
|  | { |
|  | char buf[128]; // Буфер сообщений |
|  | if (recv(fd, buf, 127, 0) > 0) |
|  | { |
|  | if (strcmp(buf, "Auction started") == 0) // Если получили сообщение о старту акциона |
|  | { |
|  | memset(buf, 0, 127); // сброс буфера |
|  | recv(fd, buf, 127, 0); // получаем время на решение |
|  | int decision\_time = atoi(buf); // переводим в инт |
|  | string result; |
|  | double given\_price; // переменна для отображения цены |
|  | window->start\_auction(decision\_time, result, given\_price); // вызываем функцию окна которая получает введённые данные и возвращает результаты |
|  | //sleep(decision\_time + 1); // ждём время на размышление чтобы предыдущая ф-ция закончилась |
|  | memset(buf, 0, 127); // сбрасываем буфер |
|  | strcpy(buf, result.c\_str()); // записываем результирующую строку в буфер |
|  | send(fd, buf, 127, 0); // отправляем результат (валюта и цена или пасс) |
|  | memset(buf, 0, 127); // сбрасываем буфер |
|  | recv(fd, buf, 127, 0); // получаем результат аукциона |
|  |  |
|  | if (buf[0] == 'T') // если мы выиграли |
|  | { |
|  | goods++; |
|  | if (result[0] == 'R') rubles -= given\_price; // определяем валюту из строки и уменьшаем нужную валюту |
|  | else if (result[0] == 'U') dollars -= given\_price; |
|  | else if (result[0] == 'E') euros -= given\_price; |
|  | } |
|  | buf[0] = ' '; // Убираем T/F из строки |
|  | window->update\_goods\_label(); // Обновляем строку кол-ва товаров в окне |
|  | window->update\_money\_label(); // Обновляем строки денег в окне |
|  | window->print\_warning\_message(buf); // выписываем результирующее сообщение в окно |
|  | } |
|  | } |
|  |  |
|  | } |
|  | } |
|  |  |
|  | int main(int argc, char \*argv[]) |
|  | { |
|  | /\* Стандартная процедура создания окна QT \*/ |
|  | QApplication a(argc, argv); |
|  |  |
|  | QTranslator translator; |
|  | const QStringList uiLanguages = QLocale::system().uiLanguages(); |
|  | for (const QString &locale : uiLanguages) { |
|  | const QString baseName = "AuctionGame\_" + QLocale(locale).name(); |
|  | if (translator.load(":/i18n/" + baseName)) { |
|  | a.installTranslator(&translator); |
|  | break; |
|  | } |
|  | } |
|  | MainWindow w; |
|  | w.show(); |
|  |  |
|  | /\* \*/ |
|  |  |
|  | in\_port\_t portno = 2400; // стандартный порт |
|  | string ip = "127.0.0.1"; // стандартный ip |
|  |  |
|  | if (argc >= 3) // если кол-во аргументов 3 и больше |
|  | { |
|  | ip = argv[1]; // ip равен аргументу 1 |
|  | portno = atoi(argv[2]); // port равен аргументу 2 |
|  | } |
|  |  |
|  | hostent \*server; |
|  | sockfd = create\_socket(); // создаём сокет |
|  | server = get\_host\_by\_ip(ip); // получаем хостинг по ip |
|  | server\_connect(portno, sockfd, server); // подключаемся к серверу по полученному хостингу |
|  |  |
|  | pthread\_t thread, auction\_thr; |
|  | pthread\_create(&thread, 0, money\_increase, 0); // Поток увеличения денег игрока |
|  | pthread\_create(&auction\_thr, NULL, auction\_handle, (void\*\*)&w); // Поток игры. Передаём QT окно в качестве аргумента |
|  |  |
|  | return a.exec(); // Запускаем QT окно |
|  | } |
|  |  |

Файл mainwindow.h

|  |
| --- |
| #ifndef MAINWINDOW\_H |
|  | #define MAINWINDOW\_H |
|  |  |
|  | #include <QMainWindow> |
|  |  |
|  | QT\_BEGIN\_NAMESPACE |
|  | namespace Ui { class MainWindow; } |
|  | QT\_END\_NAMESPACE |
|  |  |
|  | class MainWindow : public QMainWindow |
|  | { |
|  | Q\_OBJECT |
|  |  |
|  | public: |
|  | MainWindow(QWidget \*parent = nullptr); |
|  | ~MainWindow(); |
|  |  |
|  | public slots: |
|  | void start\_auction(int auction\_time, std::string& result, double& price); |
|  | void print\_warning\_message(std::string message); |
|  | void update\_money\_label(); |
|  | void update\_goods\_label(); |
|  |  |
|  | private: |
|  | Ui::MainWindow \*ui; |
|  | void start\_update\_money\_label\_thr(); |
|  |  |
|  | private slots: |
|  | void get\_ruble(); |
|  | void get\_usd(); |
|  | void get\_eur(); |
|  | void rub\_to\_usd(); |
|  | void rub\_to\_eur(); |
|  | void usd\_to\_rub(); |
|  | void usd\_to\_eur(); |
|  | void eur\_to\_rub(); |
|  | void eur\_to\_usd(); |
|  | }; |
|  | #endif // MAINWINDOW\_H |

Файл mainwindow.cpp

|  |
| --- |
| #include "mainwindow.h" |
|  | #include "ui\_mainwindow.h" |
|  | #include "unistd.h" |
|  | #include "Convert.h" |
|  | #include <QLabel> |
|  | #include <QPushButton> |
|  | #include <QLineEdit> |
|  | #include <QGridLayout> |
|  | #include <QLCDNumber> |
|  | #include <QComboBox> |
|  |  |
|  | void enable\_auc\_form(); |
|  | void disable\_auc\_form(); |
|  |  |
|  | QLabel\* rubLabel; |
|  | QLabel\* usdLabel; |
|  | QLabel\* eurLabel; |
|  | QLabel\* warningLabel; |
|  | QLineEdit\* rubConvert, \*usdConvert, \*eurConvert; |
|  |  |
|  | QGridLayout\* aucForm; |
|  | QLCDNumber\* aucTimeLeft; |
|  | QLineEdit\* aucBetValue; |
|  | QComboBox\* aucCurrency; |
|  | QLabel\* aucLabel; |
|  |  |
|  | QLabel\* goodsCntLabel; |
|  |  |
|  | MainWindow::MainWindow(QWidget \*parent) |
|  | : QMainWindow(parent) |
|  | , ui(new Ui::MainWindow) |
|  | { |
|  | ui->setupUi(this); |
|  |  |
|  | /\* Здесь получаем все элементы окна с помощью поиска по имени \*/ |
|  | rubLabel = findChild<QLabel\*>("RubleValue"); |
|  | usdLabel = findChild<QLabel\*>("UsdValue"); |
|  | eurLabel = findChild<QLabel\*>("EurValue"); |
|  |  |
|  | rubConvert = findChild<QLineEdit\*>("RubConvertCnt"); |
|  | usdConvert = findChild<QLineEdit\*>("UsdConvertCnt"); |
|  | eurConvert = findChild<QLineEdit\*>("EurConvertCnt"); |
|  |  |
|  | warningLabel = findChild<QLabel\*>("WarningLabel"); |
|  |  |
|  | aucForm = findChild<QGridLayout\*>("AucForm"); |
|  | aucTimeLeft = findChild<QLCDNumber\*>("AucTimeleft"); |
|  | aucBetValue = findChild<QLineEdit\*>("AucBetValue"); |
|  | aucCurrency = findChild<QComboBox\*>("AucCurrency"); |
|  | aucLabel = findChild<QLabel\*>("AucLabel"); |
|  | disable\_auc\_form(); |
|  |  |
|  | goodsCntLabel = findChild<QLabel\*>("GoodsCount"); |
|  |  |
|  | /\* Здесь ставим конпкам функционал \*/ |
|  | auto getRubleBtn = findChild<QPushButton\*>("GetRubleBtn"); |
|  | auto getUsdBtn = findChild<QPushButton\*>("GetUsdBtn"); |
|  | auto getEurBtn = findChild<QPushButton\*>("GetEurBtn"); |
|  | connect(getRubleBtn, SIGNAL(released()), this, SLOT(get\_ruble())); // Например кнопка получить рубиль по нажатию вызовет функцию get\_ruble() |
|  | connect(getUsdBtn, SIGNAL(released()), this, SLOT(get\_usd())); |
|  | connect(getEurBtn, SIGNAL(released()), this, SLOT(get\_eur())); |
|  |  |
|  | auto rubToUsdBtn = findChild<QPushButton\*>("RubToUsdBtn"); |
|  | auto rubToEurBtn = findChild<QPushButton\*>("RubToEurBtn"); |
|  | connect(rubToUsdBtn, SIGNAL(released()), this, SLOT(rub\_to\_usd())); |
|  | connect(rubToEurBtn, SIGNAL(released()), this, SLOT(rub\_to\_eur())); |
|  |  |
|  | auto usdToRubBtn = findChild<QPushButton\*>("UsdToRubBtn"); |
|  | auto usdToEurBtn = findChild<QPushButton\*>("UsdToEurBtn"); |
|  | connect(usdToRubBtn, SIGNAL(released()), this, SLOT(usd\_to\_rub())); |
|  | connect(usdToEurBtn, SIGNAL(released()), this, SLOT(usd\_to\_eur())); |
|  |  |
|  | auto eurToRubBtn = findChild<QPushButton\*>("EurToRubBtn"); |
|  | auto eurToUsdBtn = findChild<QPushButton\*>("EurToUsdBtn"); |
|  | connect(eurToRubBtn, SIGNAL(released()), this, SLOT(eur\_to\_rub())); |
|  | connect(eurToUsdBtn, SIGNAL(released()), this, SLOT(eur\_to\_usd())); |
|  | /\* \*/ |
|  |  |
|  | start\_update\_money\_label\_thr(); // Запускаем поток обновления строк денег |
|  | } |
|  |  |
|  | MainWindow::~MainWindow() |
|  | { |
|  | delete ui; |
|  | } |
|  |  |
|  | bool check\_rubles(double ruble) |
|  | { |
|  | if (ruble > rubles) return false; |
|  | else return true; |
|  | } |
|  |  |
|  | bool check\_usd(double usd) |
|  | { |
|  | if (usd > dollars) return false; |
|  | else return true; |
|  | } |
|  |  |
|  | bool check\_eur(double eur) |
|  | { |
|  | if (eur > euros) return false; |
|  | else return true; |
|  | } |
|  |  |
|  | // Поток одного аукциона. |
|  | void\* auction\_thread(void\* arg) |
|  | { |
|  | int auction\_time = \*(int\*)arg; // Преобразуем в int время аукциона |
|  | while (auction\_time > 0) // Пока время аукциона больше 0 |
|  | { |
|  | aucTimeLeft->display(auction\_time); // Записываем секунды в поле окна |
|  | auction\_time--; // уменьшаем |
|  | sleep(1); // ждём секунду |
|  | } |
|  |  |
|  | char\* result = new char[30]; |
|  | std::string intermediate; // Промежуточная строка для формирования результата |
|  | if (aucBetValue->text() == "") // Если в поле ставки пользователь ничего не написал |
|  | { |
|  | strcpy(result, "pas"); // то он спасовал. Копируем строку "pas" в результат |
|  | pthread\_exit(result); // выходим из потока |
|  | } |
|  | double betValue = aucBetValue->text().toDouble(); // Если пред. условие не прошло то переводим текст из поля в число Double |
|  | switch (aucCurrency->currentIndex()) // Смотрим какая валюта выбрана в поле валюты |
|  | { |
|  | case 0: // Если 0 то рубли |
|  | { |
|  | if (betValue > rubles) // Тут идут проверки что если написанная ставка больше доступных монет (например рублей 15 а игрок поставил 30 на ставку) |
|  | { |
|  | strcpy(result, "pas"); // то он пасует |
|  | pthread\_exit(result); |
|  | } |
|  | intermediate += "R"; break; // Если всё ок то добавляем в промежуточную строку символ R (рубли) |
|  | } |
|  | case 1: // 1 - доллары |
|  | { |
|  | if (betValue > dollars) // то же самое что в прошлом |
|  | { |
|  | strcpy(result, "pas"); |
|  | pthread\_exit(result); |
|  | } |
|  | intermediate += "U"; break; // доллары (usd) |
|  | } |
|  | case 2: // 2 - евро |
|  | { |
|  | if (betValue > euros) |
|  | { |
|  | strcpy(result, "pas"); |
|  | pthread\_exit(result); |
|  | } |
|  | intermediate += "E"; break; // евро (euro) |
|  | } |
|  | } |
|  | intermediate += QString::number(betValue).toStdString(); // добавляем к промежуточному результату число ставки, конвертируя его в строку |
|  | strcpy(result, intermediate.c\_str()); // копируем промежуточный результат в C строку |
|  | pthread\_exit(result); // выходим из поток с полученным результатом |
|  | } |
|  |  |
|  | // Функция запуска обработки аукциона. |
|  | void MainWindow::start\_auction(int auction\_time, std::string& result, double& price) // Вызывается из Client.cpp |
|  | { |
|  | enable\_auc\_form(); // Включаем в окне форму для ставки (поле ставки, выбор валюты и тд) |
|  | pthread\_t thread; |
|  | char\* res = NULL; |
|  | pthread\_create(&thread, 0, auction\_thread, (void\*\*)&auction\_time); // создаём поток обработки аукциона |
|  | pthread\_join(thread, (void\*\*)&res); // присоединяемся к нему |
|  | disable\_auc\_form(); // По завершению выключаем форму для ставки |
|  | result = std::string(res); // Переводим результат из потока в string |
|  | if (result != "pas") // Если не спасовал то переводим полученное строку, в которой написана ставка в число |
|  | { |
|  | std::string number\_str(result.begin() + 1, result.end()); // Здесь отбрасываем 1 букву строки обазаначающую валюту |
|  | price = std::stod(number\_str); // и число равно оставшейся строке |
|  | } |
|  | } |
|  |  |
|  | // Выводит сообщение в поле окна waringLabel |
|  | void MainWindow::print\_warning\_message(std::string message) |
|  | { |
|  | warningLabel->setText(QString::fromStdString(message)); |
|  | } |
|  |  |
|  | // Функция кнопки получить рубль. Даёт 10 рублей. |
|  | void MainWindow::get\_ruble() |
|  | { |
|  | rubles += 10; |
|  | rubLabel->setText(QString::number(rubles)); //Обновляем текст |
|  | } |
|  |  |
|  | // Функция кнопки получить доллары |
|  | void MainWindow::get\_usd() |
|  | { |
|  | dollars += 0.25; |
|  | usdLabel->setText(QString::number(dollars)); |
|  | } |
|  |  |
|  | // Функция кнопки получить евро |
|  | void MainWindow::get\_eur() |
|  | { |
|  | euros += 0.1; |
|  | eurLabel->setText(QString::number(euros)); |
|  | } |
|  |  |
|  | //// Здесь идут функции для кнопок конвертации валют |
|  | // Рубль в доллар |
|  | void MainWindow::rub\_to\_usd() |
|  | { |
|  | double rubToConv = rubConvert->text().toDouble(); // Переводим текст из поля в число |
|  | if (!check\_rubles(rubToConv)) // Вызываем функцию проверки введённых рублей |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); // Если человек написал больше чем у него есть то выводим сообщение |
|  | return; |
|  | } |
|  |  |
|  | rubles -= rubToConv; // Если всё ок то уменьшаем рубли |
|  | dollars += RubToUsd(rubToConv); // Вызываем функцию конвертации рублей в доллары из файла Convert.cpp |
|  | update\_money\_label(); // Обновляем значения валют |
|  | } |
|  |  |
|  | // Следующие функции по аналогии для каждой валюты |
|  |  |
|  | void MainWindow::rub\_to\_eur() |
|  | { |
|  | double rubToConv = rubConvert->text().toDouble(); |
|  | if (!check\_rubles(rubToConv)) |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); |
|  | return; |
|  | } |
|  |  |
|  | rubles -= rubToConv; |
|  | euros += RubToEur(rubToConv); |
|  | update\_money\_label(); |
|  | } |
|  |  |
|  | void MainWindow::usd\_to\_rub() |
|  | { |
|  | double usdToConv = usdConvert->text().toDouble(); |
|  | if (!check\_usd(usdToConv)) |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); |
|  | return; |
|  | } |
|  |  |
|  | dollars -= usdToConv; |
|  | rubles += UsdToEur(usdToConv); |
|  | update\_money\_label(); |
|  | } |
|  |  |
|  | void MainWindow::usd\_to\_eur() |
|  | { |
|  | double usdToConv = usdConvert->text().toDouble(); |
|  | if (!check\_usd(usdToConv)) |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); |
|  | return; |
|  | } |
|  |  |
|  | dollars -= usdToConv; |
|  | euros += UsdToEur(usdToConv); |
|  | update\_money\_label(); |
|  | } |
|  |  |
|  | void MainWindow::eur\_to\_rub() |
|  | { |
|  | double eurToConv = eurConvert->text().toDouble(); |
|  | if (!check\_eur(eurToConv)) |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); |
|  | return; |
|  | } |
|  |  |
|  | euros -= eurToConv; |
|  | rubles += EurToRub(eurToConv); |
|  | update\_money\_label(); |
|  | } |
|  |  |
|  | void MainWindow::eur\_to\_usd() |
|  | { |
|  | double eurToConv = eurConvert->text().toDouble(); |
|  | if (!check\_eur(eurToConv)) |
|  | { |
|  | warningLabel->setText("Недостаточно средств!"); |
|  | return; |
|  | } |
|  |  |
|  | euros -= eurToConv; |
|  | dollars += EurToRub(eurToConv); |
|  | update\_money\_label(); |
|  | } |
|  | //// |
|  |  |
|  | // Обновляет в окне строки валют по текущим валютам |
|  | void MainWindow::update\_money\_label() |
|  | { |
|  | rubLabel->setText(QString::number(rubles)); |
|  | usdLabel->setText(QString::number(dollars)); |
|  | eurLabel->setText(QString::number(euros)); |
|  | } |
|  |  |
|  | // Обновляет в окне строку количества товара |
|  | void MainWindow::update\_goods\_label() |
|  | { |
|  | goodsCntLabel->setText(QString::number(goods)); |
|  | } |
|  |  |
|  | // Поток обновления валют. Работает всегда. Таким образом Client.cpp обновляет валюты в своём файле а QT их тут прописывает. |
|  | void\* update\_money\_label\_thr(void\* arg) |
|  | { |
|  | MainWindow\* window = (MainWindow\*)arg; |
|  | while (true) |
|  | { |
|  | window->update\_money\_label(); // Просто вызываем функцию обновить валюты |
|  | sleep(5); // Каждые 5 секунд |
|  | } |
|  | } |
|  |  |
|  | // Начинает поток обновления валют |
|  | void MainWindow::start\_update\_money\_label\_thr() |
|  | { |
|  | pthread\_t thread; |
|  | pthread\_create(&thread, 0, update\_money\_label\_thr, this); |
|  | } |
|  |  |
|  | // Выключает форму ставки |
|  | void disable\_auc\_form() |
|  | { |
|  | aucLabel->setVisible(false); |
|  | aucTimeLeft->setVisible(false); |
|  | aucBetValue->setVisible(false); |
|  | aucCurrency->setVisible(false); |
|  | } |
|  |  |
|  | // Включает форму ставки |
|  | void enable\_auc\_form() |
|  | { |
|  | aucLabel->setVisible(true); |
|  | aucTimeLeft->setVisible(true); |
|  | aucBetValue->setVisible(true); |
|  | aucCurrency->setVisible(true); |
|  | } |

Файл Convert.h

|  |
| --- |
| #pragma once |
|  | #include <string> |
|  |  |
|  | extern double rubles; |
|  | extern double dollars; |
|  | extern double euros; |
|  | extern int goods; |
|  |  |
|  | extern void error(std::string msg); |
|  |  |
|  | extern double RubToUsd(double rub); |
|  |  |
|  | extern double RubToEur(double rub); |
|  |  |
|  | extern double UsdToRub(double usd); |
|  |  |
|  | extern double UsdToEur(double usd); |
|  |  |
|  | extern double EurToRub(double eur); |
|  |  |
|  | extern double EurToUsd(double eur); |

Файл Convert.cpp

|  |
| --- |
| #include "Convert.h" |
|  |  |
|  | void error(std::string msg) |
|  | { |
|  | perror(msg.c\_str()); |
|  | exit(1); |
|  | } |
|  |  |
|  | double RubToUsd(double rub) |
|  | { |
|  | return rub \* 0.014; |
|  | } |
|  |  |
|  | double RubToEur(double rub) |
|  | { |
|  | return rub \* 0.011; |
|  | } |
|  |  |
|  | double UsdToRub(double usd) |
|  | { |
|  | return usd \* 74.01; |
|  | } |
|  |  |
|  | double UsdToEur(double usd) |
|  | { |
|  | return usd \* 0.82; |
|  | } |
|  |  |
|  | double EurToRub(double eur) |
|  | { |
|  | return eur \* 89.89; |
|  | } |
|  |  |
|  | double EurToUsd(double eur) |
|  | { |
|  | return eur \* 1.21; |
|  | } |