Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционный системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе

на тему:

Программное средство «Авто-кликер»

Студент: гр. 451005

Тонко И. В.

Руководитель:

Ширай С. Ю.

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой ПОИТ

––––––––––––––––––––––––

(подпись)

––––––––––––––––– 2015 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту    Тонко Игорю Витальевичу––––––––––––––––––––––––

1. Тема работы   Программное средство  «Авто-кликер»

2. Срок сдачи студентом законченной работы––01.12.2016

3. Исходные данные к работе ———

4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке):

Введение;

1. Анализ предметной области;

2. Разработка программного средства;

3. Руководство пользователя;

Заключение.

4. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Схема алгоритма работы системы

5. Консультант по курсовой работе Ширай С. Ю.

7. Дата выдачи задания 25.02.2016 г.–––––––––––––––––––––––   –

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):

раздел 1,2 к 20.03.2016 – 15 % готовности работы;

разделы 3, 4 к 15.04.2016 – 30 % готовности работы;

разделы 5, 6, к 07.05.2016 – 60 % готовности работы;

раздел 7, 8, 9 к 10.05.2016 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 10.12.2015- 100 % готовности работы.

Защита курсового проекта с 12 мая 2016.

РУКОВОДИТЕЛЬ - Ширай С. Ю.

(подпись)

Задание принял к исполнению –––\_\_\_\_–– И. В. Тонко    25.02.2016 г.

(дата и подпись студента)

**СОДЕРЖАНИЕ**

Введение 4

1 Анализ предметной области 5

1.1 Анализ аналогов программного средства 5

1.2 Постановка задачи 5

2 Разработка программного средства 6

2.1 Общая информация 6

2.2 Написание кода 6

2.2.1 Сервер 6

2.2.2 Библиотека классов TonkoinLibrary 7

2.2.3 Взаимодействие клиентов 8

2.2.4 Работа с транзакциями 10

2.2.5 Эмиссия 12

2.3 Визуальное представление 12

2.3.1 Главное окно 12

2.3.2 Окно регистрации 13

2.3.3 Окно формирования транзакций 14

3 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА 15

4 Руководство пользователя 16

4.1 Начало работы 16

4.2 Авторизация 16

4.3 Осуществление транзакции 16

Заключение 17

Список использованных источников 18

Приложение А. Исходный код программы 19

Приложение Б. Блок-схема главного цикла программы 47

**ВВЕДЕНИЕ**

В современном мире наиболее развивающаяся отрасль – это отрасль компьютеров. Еще несколько десятилетий назад все персональные компьютеры можно было пересчитать по пальцам. Однако, сейчас их количество возросло настолько, что в каждой семье есть один, а может быть и более компьютеров. В наше время информационные технологии занимают центральное место в интеллектуализации общества, развитии его системы образования и культуры.

Одной из самых перспективных областей компьютерных технологий являются компьютерные сети. Компьютерные сети могут рассматриваться как средство передачи информации на большие расстояния, для чего в них применяются методы кодирования и мультиплексирования данных, получившие развитие в различных телекоммуникационных системах.

В настоящее время широкое распространение получили одноранговые сети, использующие технологию блокчейн. Блокчейн подобен коллективному разуму, которым никто не управляет, но каждый может заглянуть в его содержимое. Главные достоинства технологии – децентрализованность, публичность и абсолютная прозрачность. Блокчейн позволяет строить отношения с организациями и людьми, работа которых не вызывает у вас доверия, но оставаться уверенным, что вы защищены от любого обмана. Эта технология не дает возможности подделывать документы, изменять уже заверенную информацию или злоупотреблять служебным положением. Сфера применения блокчейна почти безгранична: от банков и рынка недвижимости до сельского хозяйства и госуправления.

1. Анализ предметной области
   1. Анализ аналогов программного средства

**«**Bitcoin**»** – самая распространенная и успешная криптовалюта. Bitcoin появился в 2008 году под авторством [Сатоши Накамото](http://ru.bitcoinwiki.org/%D0%A1%D0%B0%D1%82%D0%BE%D1%88%D0%B8_%D0%9D%D0%B0%D0%BA%D0%B0%D0%BC%D0%BE%D1%82%D0%BE).

Bitcoin, его транзакции и эмиссию регулирует обширная peer-to-peer сеть. Bitcoin использует распределенную в сети единую базу данных, включенную в децентрализованную одноранговую сеть, которая использует электронную цифровую подпись и поддерживается [proof-of-work](http://ru.bitcoinwiki.org/Proof-of-work) протоколом для обеспечения безопасности и лигитимности представленных в оборот средств.

Bitcoin является наиболее распространенной альтернативной валютой, общая рыночная стоимость которой составляет свыше 11 млрд. долларов. Возможность обменивать, продавать или покупать Bitcoin предоставлена на множестве сайтов.

Несмотря на то, что формально для использования Bitcoin не требуется идентификация личности пользователя, валюта не является полностью анонимной.

Благодаря тому, что исходный код Bitcoin находится в общем доступе, на его основе появилось множество криптовалют. Одна из них, Litecoin, использует более сложную операцию доказательства выполнения работы.

* 1. Постановка задачи

Задача данной курсовой работы заключается в разработке приложения для совершения транзакций в одноранговой сети с использованием технологии блокчейн.

Необходимо реализовать взаимодействие клиентов между собой. Каждый пользователь должен иметь возможность формировать транзакции, которые должны быть проверены остальными пользователями. Решение о принятии транзакции принимать, если транзакция принята более чем 50% пользователей.

Транзакции хранить в виде цепочки блоков, содержащих несколько транзакций в каждом.

1. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА
   1. Общая информация

Для создания данного приложения используется язык программирования С++ и интегрированная среда разработки Microsoft Visual Studio 2015. Данный продукт был выбран, так как он позволяет разрабатывать приложения под операционную систему Windows.

Приложение написано в процедурном стиле.

При проектировании программы использовалась методика нисходящего проектирования: сначала прорабатывается основная концепция программы, затем детально продумывается каждая отдельная функциональная составляющая.

* 1. Написание кода
     1. Сервер

Несмотря на все достоинства децентрализованных систем, они имеют серьезную проблему: определение адресов клиентов. Решением этой проблемы был выбран сервер, единственной функцией которого является формирование и распространение списка адресов активных клиентов.

При авторизации пользователя автоматически отправляется запрос на сервер. Принимая запрос, сервер добавляет нового клиента в список и высылает обратно полный список клиентов в сети на данный момент.

При завершении работы приложения на сервер посылается запрос на удаление клиента из списка.

Сам сервер представляет собой консольное приложение.

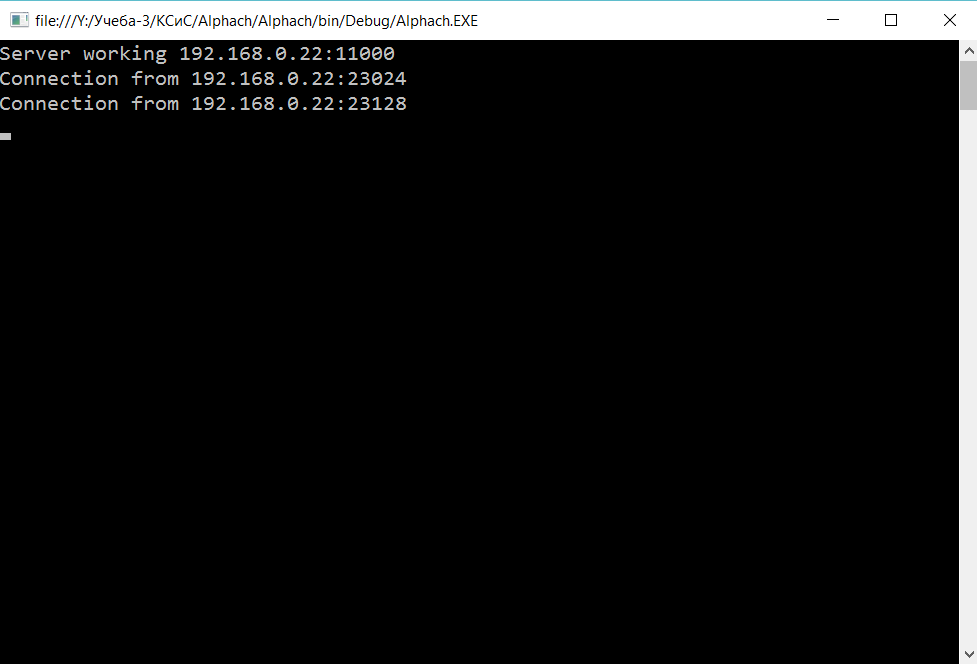


Рисунок 2.4 - Сервер

Клиент представляется классом UserIP. Класс располагает двумя свойствами: ID - логин пользователя и IP - IP-адресс пользовтеля.

* + 1. Библиотека классов TonkoinLibrary

Классы, используемые для реализации технологии блокчейн, вынесены в отдельную библиотеку классов TonkoinLibrary.

Для передачи данных по сети используется бинарная сериализация, реализуемая с помощью класса BinaryFormatter, представленным библиотекой System.Runtime.Serialization.Formatters.Binary. Использование отдельной от клиента библиотеки классов позволяет корректно десериализовать данные, сериализованные другим пользователем.

Объект класса User содержит в себе логин пользователя, hash-значение логина, hash-значение пароля и hash-значение логина пользователя, пригласившего в систему. Конструктор класса принимает три параметра типа string: логин, пароль и логин пригласившего пользователя. Далее конструктор получает и сохраняет hash-значения.

Объект класса Block представляет собой единицу цепочки блоков - главного принципа технологии Блокчейн. Блок хранит список транзакций, hash-значение предыдущего блока и свое hash-значение, которое пересчитывается каждый раз при добавлении транзакции к списку. Когда злоумышленник изменяет транзакции в блоке, это влияет на hash всего блока, который будет прежним у большинства других пользователей, поэтому такой блок не будет принят. Также это позволяет проверять список всех транзакций, проверяя hash-значения блоков, а не сравнивая весь список, размер которого быстро увеличивается.

Объект класса Transaction представляет собой транзакцию. Каждая транзакция имеет список входов, список выходов и свое hash-значение, которое формируется конструктором класса.

Объект класса In представляет собой единицу списка входов транзакции. Вход - это транзакция, из которой берутся деньги для использования в новой транзакции. В каждом входе указывается hash-значение используемой транзакции, номер выхода, с которого необходимо взять деньги и подпись владельца - hash-значение пароля пользователя-владельца транзакции.

Объект класса Out представляет собой единицу списка выходов. Выходов у транзакции может быть один или два. Если сумма денег списка входов превышает сумму для передачи другому пользователю, остаток возвращается отправителю в виде второго выхода транзакции. На выходе указывается hash-значение логина получателя и количество получаемых денег.

* + 1. Взаимодействие клиентов

После авторизации пользователя и успешного соединения с сервером, клиент запускает "слушателя" входящих пакетов в отдельном потоке. Такой "слушатель" представлен объектом класса Listener.

Открытый для прослушивания сокет связывается с портом и входит в бесконечный цикл.

IPEndPoint ipEndPoint = new IPEndPoint(userIP.IP, 12000);

Socket sListener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

try

{

sListener.Bind(ipEndPoint);

sListener.Listen(10);

}

}

catch(Exception e)

{

ExceptionReceived(this, e);

return;

}

При принятии подключения другого клиента, осуществляется прием данных и запись в объект класса MemoryStream.

Socket handler = sListener.Accept();

MemoryStream ms = new MemoryStream();

while ((bytesRec = handler.Receive(bytes)) != 0)

{

ms.Write(bytes, 0, bytesRec);

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

bytes[i] = 0;

total += bytesRec;

}

Первый байт полученных данных определяет, каким образом эти данные следует обрабатывать. Список заголовков:

* USERLIST: заголовок говорит о том, что принятый пакет данных представляет собой список пользователей, и должен быть записан в файл.
* ASKTRANSACTION: запрос на проверку транзакции, присылаемой в пакете данных. Клиент, получивший такой запрос, выполняет проверку транзакции и, в случае подтверждения, высылает ответ, что транзакция справедлива.
* VALIDTRANSACTION: получение такого пакета говорит о том, что клиент, получивший от текущего пользователя запрос на проверку транзакции ASKTRANSACTION, подтвердил справедливость новой транзакции. Если транзакция подтверждена более чем 51% пользователей, то она добавляется в список и всем пользователям высылается запрос ADDTRANSACTION.
* ADDTRANSACTION: транзакция, следуемая за таким заголовком, является проверенной и подлежит добавлению в список.
* ASKBLOCKLIST: запрос на проверку списка блоков. За таким заголовком идет список hash-значений блоков клиента-отправителя. Они сравниваются с hash-значениями блоков получателя и в случае соответствия, высылается ответ с одобрением списка.
* VALIDBLOCKLIST: положительный ответ на проверку списка блоков. Если список блоков получает одобрение более 51%   
  пользователей, он считается справедливым и может использоваться для работы клиента.

Отправка пакетов осуществляется экземпляром класса BinSender, который в своем конструкторе выполняет запрос серверу и получает актуальный список пользователей в сети. Пакет формируется с помощью соответствующих методов. Отправка запросов всем клиентам осуществляется с помощью метода SendBin.

public void SendBin(byte[] b)

{

foreach (UserIP userIP in storage.userIPList)

{

IPEndPoint ipEndPoint = new IPEndPoint(userIP.IP, 12000);

try

{

Socket sender = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

if (userIP.IP.ToString() != storage.curentUserIP.IP.ToString())

{

sender.Connect(ipEndPoint);

int bytesSent = sender.Send(b);

sender.Shutdown(SocketShutdown.Both);

}

sender.Close();

}

catch(Exception e)

{

ExceptionReceived(this, e);

return;

}

}

}

Ответы на запросы отправляются аналогичным образом, за исключением того, что ответ отправляется только тому клиенту, который отправил запрос, а не всем активным пользователям.

* + 1. Работа с транзакциями

Большинство функций, используемых для работы с транзакциями, реализованы методами класса TransactionVerifier:

* TransactionUsed: выполняет проверку, были ли использованы выходы данной транзакции, или нет. Возвращает 0, если ни один выход не был использован, 1, если был использован первый выход, 2 - если второй и 3 - если оба выхода были использованы.
* IsTransactionValid: выполняет формирование новой транзакции, осуществляет поиск средств и в случае успеха возвращает новую транзакцию. В случае неудачи возвращает null.
* IsTransactionValid: выполняет проверку уже существующей транзакции. Возвращает true, если транзакция справедлива, и false в противном случае.
* ShowBallance: осуществляет поиск всех неиспользованных выходов, принадлежащих пользователю, и выдает результат, который соответствует сумме средств, которыми располагает пользователь.
* PackBlock: вычисляет hash-значение блока на основе hash-значений транзакций этого блока.
* FindParent: возвращает hash-значение логина пользователя, пригласившего текущего пользователя.
* BonusTonkoin: создание эмиссионных транзакций в начале каждого блока.
* SearchSender: возвращает hash-значение логина отправителя заданной в параметрах транзакции.

Процесс осуществления транзакции:

1. Пользователь создает транзакцию, используя окно формирования транзакций.
2. Если транзакция предназначена существующему пользователю и у отправителя достаточно средств, создается объект класса Transaction.
3. Транзакция отправляется всем активным пользователям в сети с заголовком ASKTRANSACTION.
4. Остальные пользователи проверяют транзакцию на своей стороне и отправляют ответ.
5. Когда отправитель получает одобрение более чем половины пользователей, он добавляет транзакцию в список и отправляет её всем пользователям с заголовком ADDTRANSACTION.
6. Каждый пользователь добавляет транзакцию к списку на своей стороне.

Перед тем, как загрузить список транзакций, необходимо удостовериться в том, что список достоверен. С этой целью проводится проверка цепочки блоков после авторизации клиента.

Процесс проверки цепочки блоков:

1. Клиент открывает файл с блоками, повторно высчитывает hash-значения каждого блока.
2. Все hash-значения блоков собираются в правильном порядке и передаются другим пользователям под заголовком ASKBLOCKLIST.
3. Принимая такой заголовок, клиент также открывает файл с цепочкой блоков и сравнивает hash-значения блоков.
4. При совпадении всех значений, высылается ответ с заголовком VALIDBLOCKLIST.
5. При получении одобрений более чем от половины пользователей, список загружается и становится доступным для работы.
   * 1. Эмиссия

Эмиссия почти тем же способом, как это сделано в Bitcoin: бонусные деньги получает пользователь, создающий новый блок.

В программном средстве Tonkion эмиссия дополнена: кроме пользователя, создавшего новый блок, бонусные деньги (в меньшем размере) получает пользователь, пригласивший создающего новый блок пользователя в систему. Таким образом, первый пользователь в системе получает бонус каждый раз. Такая система стимулирует пользователей приглашать в систему как можно большее число новых пользователей.

* 1. Визуальное представление

Для взаимодействия приложения с пользователем создаются три окна:

* главное окно с полем для авторизации, функциональными кнопками и компонентом для вывода информации;
* окно для регистрации новых пользователей;
* окно для формирования новой транзакции.
  + 1. Главное окно

Главное окно открывается при запуске приложения. Если у пользователя есть учетная запись в системе, то он может ввести свой логин и пароль и войти, нажав кнопку Enter.

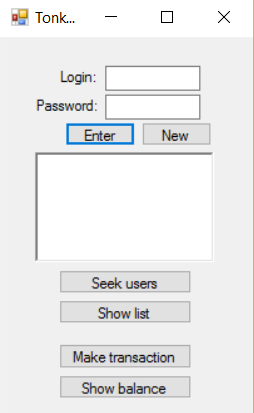


Рисунок 2.1 - Главное окно

Кнопка New открывает окно для регистрации.

Кнопка Seek users позволяет обновить список пользователей, находящихся в сети.

Кнопка Show list выводит на экран список всех зарегистрированных пользователей.

Кнопка Make transaction открывает окно для составления транзакции.

Кнопка Show balance отображает количество валюты, принадлежащей текущему пользователю.

* + 1. Окно регистрации

Чтобы добавить новую учетную запись, необходимо указать логин, пароль, подтвердить пароль и указать логин пользователя, пригласившего нового пользователя в систему.

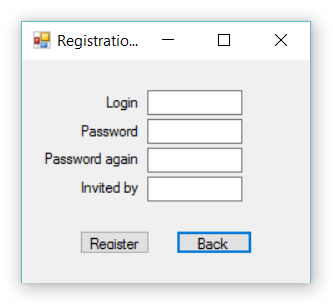


Рисунок 2.2 - Окно регистрации

* + 1. Окно формирования транзакций

Для формирования новой транзакции необходимо указать получателя, количество переводимых средств и пароль отправителя.

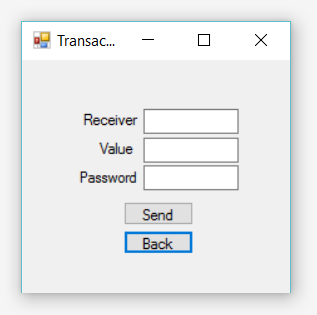


Рисунок 2.3 - Окно формирования транзакций

1. ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

После написания исходного кода проводились испытания программного средства. Во время тестирования проверялась работа системы

с тремя активными пользователями.

Все места, в которых наиболее вероятно возникновение ошибки, выполнялись по шагам.

Была выявлена ошибка с вычислением hash-значений, которые оказывались различными при вызове функции с одинаковыми параметрами. Ошибка заключалась в том, что после вычисления hash-значения необходимо заново инициализировать используемые для вычисления данные.

Была выявлена ошибка с десериализацией полученных по сети данных.

Ошибка заключалась в некорректном приеме данных, поэтому прием данных принял вид следующего цикла:

MemoryStream ms = new MemoryStream();

while ((bytesRec = handler.Receive(bytes)) != 0)

{

ms.Write(bytes, 0, bytesRec);

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

bytes[i] = 0;

total += bytesRec;

}

Все найденные ошибки функционала приложения исправлялись по мере нахождения. После внесения исправлений проводились повторные тесты.

1. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ
   1. Начало работы

Перед началом работы необходимо запустить сервер. После двойного щелчка по иконке приложения откроется окно программы.

* 1. Авторизация

Пользователи, имеющие учетную запись, могут войти, используя свой логин и пароль. Если учетной записи нет, следует создать новую, открыв окно для регистрации нажав кнопку New.

В открывшейся форме необходимо ввести логин, пароль, подтвердить пароль. Также, необходимо указать пользователя, который пригласил нового пользователя. Если такого пользователя нет, следует выбрать любого из списка уже зарегистрированных.

* 1. Осуществление транзакции

Для осуществлений транзакции следует открыть окно формирования транзакций, нажав кнопку Make transaction. Далее необходимо указать получателя, сумму перевода и пароль.

Если такая транзакция возможна, пользователь увидит сообщение “Success”. После этого транзакция будет проверена другими пользователями и, в случае принятия, выведется сообщение “Transaction accepted”.

**ЗАКЛЮЧЕНИЕ**

Результатом данной курсовой работы является программное средство Tonkoin, полностью удовлетворяющее всем условиям поставленной задачи. Данное приложение способно работать в локальной сети и со множеством клиентов одновременно.

В ходе разработки было изучено большое количество литературы из различных источников. На примере Bitcoin, был изучен принцип формирования транзакции и построения цепочки блоков.

Проделанная работа позволяет в полной мере раскрыть у автора наличие базовых знаний и основных навыков в области сетевого программирования.

Программное средство Tonkoin можно продолжать развивать, переходя от локальной сети к глобальной. Также со временем будет изменен механизм эмиссии и будет добавлена возможность восстанавливать поврежденные файлы, содержащие список блоков транзакций.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Руководство по программированию на C# [Электронный ресурс] - Режим доступа : https://msdn.microsoft.com. Дата доступа : 15.05.2016
2. Bitcoin. Как это работает [Электронный ресурс] - Режим доступа: https://habrahabr.ru/post/114642/ . Дата доступа: 15.05.2016
3. Мартин.Р., Чистый код. Создание, анализ и рефакторинг: Петербург,2015. –464 с.

Приложение А. Исходный код программы

Проект tonkoin

Класс Program

using System;

using System.Windows.Forms;

namespace tonkoin

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Tonkoin());

}

}

}

Класс Tonkoin

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using tonkoin.Serializers;

using TonkoinLibrary;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

public partial class Tonkoin : Form

{

public Storage storage = new Storage();

bool listenerEnabled = false;

String from;

int value;

public Tonkoin()

{

InitializeComponent();

InitializeStorage();

}

private void InitializeStorage()

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += OnExceptionReceived;

storage.userList = bs.GetUserList();

storage.newTransaction = null;

storage.Verified = 0;

}

private void enterBT\_Click(object sender, EventArgs e)

{

String login = loginTB.Text;

if (login == "")

{

MessageBox.Show("Please, type your login.");

return;

}

String password = passwordTB.Text;

if (login == "")

{

MessageBox.Show("Please, type your password.");

return;

}

Sha256Digest sh = new Sha256Digest();

byte[] s = Encoding.UTF8.GetBytes(password);

uint[] hash = sh.hash(s);

if (storage.userList != null)

{

foreach (User user in storage.userList)

{

if (user.ID == login)

{

if (sh.IsEqualHash(hash, user.password\_hash))

{

storage.curentUser = user;

MessageBox.Show("Enter success.");

}

else

{

MessageBox.Show("Wrong password.");

return;

}

}

}

if (storage.curentUser == null)

MessageBox.Show("User " + login + " not found.");

SeekUsers();

BinSender bsend = new BinSender(storage);

bsend.ExceptionReceived += OnExceptionReceived;

while (!listenerEnabled) { }

storage.Verified = 1;

bsend.VerifyBlockList();

}

else

{

MessageBox.Show("Failed to enter. User list not found.");

}

}

private void DisplayList(List<UserIP> userIpList)

{

onlineRTB.Text = "";

foreach (UserIP us in userIpList)

{

onlineRTB.Text += us.ID + ": " + us.IP.ToString() + '\n';

}

}

private void DisplayList(List<User> usepList)

{

onlineRTB.Text = "";

foreach (User us in storage.userList)

{

onlineRTB.Text += us.ID + '\n';

}

}

private void OnTransactionReceived(object sender, Transaction transaction)

{

MessageBox.Show("Transaction accepted.");

}

private void OnExceptionReceived(object sender, Exception e)

{

MessageBox.Show(e.ToString());

}

private void SeekUsers()

{

UserSeeker userSeeker = new UserSeeker();

userSeeker.ExceptionReceived += OnExceptionReceived;

storage.curentUserIP = new UserIP();

if (storage.curentUser != null)

{

storage.curentUserIP.ID = storage.curentUser.ID;

}

else

storage.curentUserIP.ID = "Guest";

storage.userIPList = userSeeker.GetUserList(storage.curentUserIP);

if (storage.userIPList != null)

{

foreach (UserIP userIP in storage.userIPList)

{

if (userIP.ID == storage.curentUserIP.ID)

{

storage.curentUserIP.IP = userIP.IP;

break;

}

}

DisplayList(storage.userIPList);

if (!listenerEnabled)

{

listenerEnabled = true;

Task.Factory.StartNew(() =>

{

var listener = new Listener(storage);

listener.TransactionReceived += OnTransactionReceived;

listener.ExceptionReceived += OnExceptionReceived;

listener.StartListen(storage.curentUserIP);

});

}

}

}

private void seekBT\_Click(object sender, EventArgs e)

{

SeekUsers();

}

private void showUsersBT\_Click(object sender, EventArgs e)

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += OnExceptionReceived;

storage.userList = bs.GetUserList();

DisplayList(storage.userList);

}

private void button1\_Click(object sender, EventArgs e)

{

if (storage.userList != null)

{

RegistrationForm rf = new RegistrationForm();

rf.Owner = this;

rf.Show();

}

}

private void Tonkoin\_FormClosing(object sender, FormClosingEventArgs e)

{

UserSeeker userSeeker = new UserSeeker();

userSeeker.ExceptionReceived += OnExceptionReceived;

userSeeker.UserExit();

}

private void updateListBT\_Click(object sender, EventArgs e)

{

BinSender binSender = new BinSender(storage);

binSender.ExceptionReceived += OnExceptionReceived;

binSender.SendUserList();

}

private void makeTransBT\_Click(object sender, EventArgs e)

{

if (storage.curentUser != null)

{

TransactionForm tf = new TransactionForm();

tf.Owner = this;

tf.Show();

}

else

MessageBox.Show("Please, enter first");

}

private void showBalanceBT\_Click(object sender, EventArgs e)

{

if (storage.curentUser != null)

{

if (storage.blockList == null)

{

MessageBox.Show("Please, try again.");

BinSender bs = new BinSender(storage);

bs.ExceptionReceived += OnExceptionReceived;

BinSerializer bser = new BinSerializer();

bs.ExceptionReceived += OnExceptionReceived;

bs.VerifyBlockList();

}

if (storage.blockList != null)

{

if (storage.curentUser != null)

{

TransactionVerifier tr = new TransactionVerifier();

MessageBox.Show("You have " + tr.ShowBallance(storage).ToString() + " tonkoins.");

}

}

}

else

MessageBox.Show("Please, enter.");

}

private void getBlockListBT\_Click(object sender, EventArgs e)

{

}

}

}

Класс Storage

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

public class Storage

{

public List<UserIP> userIPList { get; set; }

public List<User> userList { get; set; }

public List<Block> blockList { get; set; }

public UserIP curentUserIP { get; set; }

public User curentUser { get; set; }

public Transaction newTransaction { get; set; }

public int Verified { get; set; }

public Storage() { }

}

}

Класс UserSeeker

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

namespace tonkoin

{

class UserSeeker

{

public UserSeeker() { }

public event EventHandler<Exception> ExceptionReceived;

public List<UserIP> GetUserList(UserIP curUser)

{

IPEndPoint ipEndPoint = new IPEndPoint(IPAddress.Parse("192.168.43.89"), 11000);

Socket sender = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

byte[] bytesOut = new byte[2000];

byte[] bytesIn = new byte[2000];

try

{

sender.Connect(ipEndPoint);

curUser.IP = ((IPEndPoint)sender.LocalEndPoint).Address;

List<UserIP> userIPList = new List<UserIP>();

userIPList.Add(curUser);

UserIPSerializer s = new UserIPSerializer();

bytesOut = s.Serialize(userIPList);

int bytesSent = sender.Send(bytesOut);

int bytesRec = sender.Receive(bytesIn);

userIPList = s.Deserialize(bytesIn);

sender.Shutdown(SocketShutdown.Both);

sender.Close();

return userIPList;

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public void UserExit()

{

IPEndPoint ipEndPoint = new IPEndPoint(IPAddress.Parse("192.168.43.89"), 11000);

Socket sender = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

try

{

sender.Connect(ipEndPoint);

byte[] bytesOut = new byte[1];

bytesOut[0] = 0;

int bytesSent = sender.Send(bytesOut);

sender.Shutdown(SocketShutdown.Both);

sender.Close();

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

}

}

Класс BinSerializer

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.IO;

using System.Runtime.Serialization.Formatters.Binary;

using TonkoinLibrary;

using TonkoinLibrary.Transactions;

namespace tonkoin.Serializers

{

class BinSerializer

{

public event EventHandler<Exception> ExceptionReceived;

private BinaryFormatter binFormatter;

public BinSerializer()

{

binFormatter = new BinaryFormatter();

}

public void Seralize(List<User> unitList, Stream target)

{

try

{

binFormatter.Serialize(target, unitList);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

public void Seralize(Transaction transaction, Stream target)

{

try

{

binFormatter.Serialize(target, transaction);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

public void Seralize(List<Block> blockList, Stream target)

{

try

{

binFormatter.Serialize(target, blockList);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

public List<User> DeserializeUnit(Stream target)

{

try

{

return (List<User>)binFormatter.Deserialize(target);

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public List<Block> DeserializeBlock(Stream target)

{

try

{

return (List<Block>)binFormatter.Deserialize(target);

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public Transaction DeserializeTransaction(Stream target)

{

try

{

return (Transaction)binFormatter.Deserialize(target);

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public List<User> GetUserList()

{

try

{

Stream stream\_target = File.Open("users.bin", FileMode.Open, FileAccess.Read, FileShare.None);

List<User> userList = new List<User>();

userList = this.DeserializeUnit(stream\_target);

stream\_target.Close();

return userList;

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public void SaveUserList(List<User> userList)

{

Stream stream\_target = File.Open("users.bin", FileMode.Create, FileAccess.Write, FileShare.None);

StreamWriter streamW = new StreamWriter(stream\_target);

this.Seralize(userList, stream\_target);

stream\_target.Close();

}

public List<Block> GetBlockList()

{

try

{

Stream stream\_target = File.Open("blockChain.bin", FileMode.Open, FileAccess.Read, FileShare.None);

List<Block> blockList = new List<Block>();

blockList = this.DeserializeBlock(stream\_target);

stream\_target.Close();

TransactionVerifier tv = new TransactionVerifier();

foreach(Block bl in blockList)

{

bl.hash = tv.PackBlock(bl); }

return blockList;

}

catch (Exception e)

{

ExceptionReceived(this, e);

return null;

}

}

public void SaveBlockList(List<Block> blockList)

{

Stream stream\_target = File.Open("blockChain.bin", FileMode.Create, FileAccess.Write, FileShare.None);

StreamWriter streamW = new StreamWriter(stream\_target);

this.Seralize(blockList, stream\_target);

stream\_target.Close();

}

}

}

Класс BinSender

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using tonkoin.Serializers;

using System.IO;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

class BinSender

{

public event EventHandler<Exception> ExceptionReceived;

enum Header

{

USERLIST = 1, TRANSLIST = 2, BLOCKLIST = 3, ASKTRANSACTION = 4, VALIDTRANSACTION = 5,

ASKBLOCKLIST = 6, VALIDBLOCKLIST = 7, NOTVALIDBLOCKLIST = 8, ADDTRANSACTION = 9

}

public Storage storage;

public BinSender(Storage storage)

{

this.storage = storage;

UserSeeker us = new UserSeeker();

us.ExceptionReceived += ExceptionReceived;

this.storage.userIPList = us.GetUserList(storage.curentUserIP);

}

public void SendBin(byte[] b)

{

foreach (UserIP userIP in storage.userIPList)

{

IPEndPoint ipEndPoint = new IPEndPoint(userIP.IP, 12000);

try

{

Socket sender = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

if (userIP.IP.ToString() != storage.curentUserIP.IP.ToString())

{

sender.Connect(ipEndPoint);

int bytesSent = sender.Send(b);

sender.Shutdown(SocketShutdown.Both);

}

sender.Close();

}

catch (Exception e)

{

ExceptionReceived(this, e);

continue;

}

}

}

public void SendUserList()

{

try

{

Stream stream\_target = File.Open("users.bin", FileMode.Open, FileAccess.Read, FileShare.None);

byte[] b = new byte[(int)stream\_target.Length + 1];

b[0] = (byte)Header.USERLIST;

stream\_target.Read(b, 1, (int)stream\_target.Length);

stream\_target.Close();

SendBin(b);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

public void SendTransaction(List<UserIP> userIPList, UserIP curIP, Transaction transaction)

{

MemoryStream ms = new MemoryStream();

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

bs.Seralize(transaction, ms);

ms.Seek(0, SeekOrigin.Begin);

byte[] b = new byte[ms.Length + 1];

b[0] = (byte)Header.ASKTRANSACTION;

ms.Read(b, 1, (int)ms.Length);

ms.Close();

storage.newTransaction = transaction;

storage.Verified = 1;

SendBin(b);

}

public void SendValid(IPAddress ip, byte[] b)

{

try

{

IPEndPoint ipEndPoint = new IPEndPoint(ip, 12000);

Socket sender = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

sender.Connect(ipEndPoint);

int bytesSent = sender.Send(b);

sender.Shutdown(SocketShutdown.Both);

sender.Close();

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

public void VerifyBlockList()

{

try

{

Stream blockFile = File.Open("blockChain.bin", FileMode.Open, FileAccess.Read, FileShare.None);

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

TransactionVerifier tv = new TransactionVerifier();

List<Block> blockList = bs.DeserializeBlock(blockFile);

byte[] buffer = new byte[4];

byte[] b = new byte[blockList.Count \* 32 + 1];

for (int i = 0; i < blockList.Count; i++)

{

blockList.ElementAt(i).hash = tv.PackBlock(blockList.ElementAt(i));

for (int j = 0; j < 8; j++)

{

buffer = BitConverter.GetBytes(blockList.ElementAt(i).hash[j]);

for (int k = 0; k < 4; k++)

{

b[i \* 32 + j \* 4 + k + 1] = buffer[k];

}

}

}

b[0] = (byte)Header.ASKBLOCKLIST;

blockFile.Close();

SendBin(b);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

}

}

}

Класс Listener

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

using System.Net.Sockets;

using tonkoin.Serializers;

using System.IO;

using TonkoinLibrary.Transactions;

using System.Threading;

namespace tonkoin

{

class Listener

{

enum Header { USERLIST = 1, TRANSLIST = 2, BLOCKLIST = 3, ASKTRANSACTION = 4, VALIDTRANSACTION = 5,

ASKBLOCKLIST = 6, VALIDBLOCKLIST = 7, NOTVALIDBLOCKLIST = 8, ADDTRANSACTION = 9}

public event EventHandler<Transaction> TransactionReceived;

public event EventHandler<Exception> ExceptionReceived;

Storage storage;

public Listener(Storage st)

{

this.storage = st;

}

public void StartListen(UserIP userIP)

{

IPEndPoint ipEndPoint = new IPEndPoint(userIP.IP, 12000);

Socket sListener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

try

{

sListener.Bind(ipEndPoint);

sListener.Listen(10);

}

catch(Exception e)

{

ExceptionReceived(this, e);

return;

}

while (true)

{

byte[] bytes = new byte[65536];

int bytesRec = 0;

int total = 0;

Socket handler = sListener.Accept();

MemoryStream ms = new MemoryStream();

while ((bytesRec = handler.Receive(bytes)) != 0)

{

ms.Write(bytes, 0, bytesRec);

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

bytes[i] = 0;

total += bytesRec;

}

Header byteHeader;

ms.Seek(0, SeekOrigin.Begin);

byteHeader = (Header)ms.ReadByte();

switch (byteHeader)

{

case Header.USERLIST:

{

try

{

Stream stream\_target = File.Open("users.bin", FileMode.Create, FileAccess.Write, FileShare.None);

ms.Position = 1;

ms.CopyTo(stream\_target);

stream\_target.Close();

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

break;

}

case Header.ASKTRANSACTION:

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

Transaction tr;

ms.Seek(1, SeekOrigin.Begin);

tr = bs.DeserializeTransaction(ms);

TransactionVerifier trv = new TransactionVerifier();

if (trv.IsTransactionValid(storage, tr))

{

byte[] sendB = new byte[ms.Length];

ms.Seek(0, SeekOrigin.Begin);

ms.Read(sendB, 0, (int)ms.Length);

sendB[0] = (byte)Header.VALIDTRANSACTION;

BinSender bsend = new BinSender(storage);

bsend.ExceptionReceived += ExceptionReceived;

bsend.SendValid(((IPEndPoint)handler.RemoteEndPoint).Address, sendB);

}

break;

}

case Header.VALIDTRANSACTION:

{

if (storage.newTransaction != null)

{

storage.Verified++;

if (storage.Verified > (storage.userIPList.Count / 2))

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

Transaction tr;

ms.Seek(1, SeekOrigin.Begin);

tr = bs.DeserializeTransaction(ms);

Sha256Digest sha = new Sha256Digest();

if (sha.IsEqualHash(tr.hash, storage.newTransaction.hash))

{

TransactionVerifier tv = new TransactionVerifier();

storage.blockList.Last().trans\_list.Add(storage.newTransaction);

storage.blockList.Last().hash = tv.PackBlock(storage.blockList.Last());

if (storage.blockList.Last().trans\_list.Count == 10)

{

storage.blockList.Last().hash = tv.PackBlock(storage.blockList.Last());

Block block = new Block();

block.hash\_before = storage.blockList.Last().hash;

block.trans\_list = new List<Transaction>();

uint[] rec = tv.SearchSender(storage, tr.in\_list.ElementAt(0).hash, tr.in\_list.ElementAt(0).n);

storage.blockList.Add(block);

tv.BonusTonkoin(storage, rec);

block.hash = tv.PackBlock(block);

}

bs.SaveBlockList(storage.blockList);

storage.newTransaction = null;

BinSender bsend = new BinSender(storage);

bsend.ExceptionReceived += ExceptionReceived;

byte[] b = new byte[ms.Length];

ms.Seek(0, SeekOrigin.Begin);

ms.Read(b, 0, (int)ms.Length);

b[0] = (byte)Header.ADDTRANSACTION;

bsend.SendBin(b);

}

}

}

break;

}

case Header.ASKBLOCKLIST:

{

try

{

Stream blockFile = File.Open("blockChain.bin", FileMode.Open, FileAccess.Read, FileShare.None);

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

List<Block> blockList = bs.DeserializeBlock(blockFile);

ms.Seek(1, SeekOrigin.Begin);

TransactionVerifier tv = new TransactionVerifier();

int i = 0;

byte[] buffer = new byte[4];

byte[] b = new byte[blockList.Count \* 32];

for (i = 0; i < blockList.Count; i++)

{

blockList.ElementAt(i).hash = tv.PackBlock(blockList.ElementAt(i));

for (int j = 0; j < 8; j++)

{

buffer = BitConverter.GetBytes(blockList.ElementAt(i).hash[j]);

for (int k = 0; k < 4; k++)

{

b[i \* 32 + j \* 4 + k] = buffer[k];

}

}

}

i = 0;

if ((ms.Length - 1) == (b.Length))

while ((i < b.Length) && (ms.ReadByte() == b[i]))

i++;

byte[] sendB = new byte[1];

if (i == b.Length)

sendB[0] = (byte)Header.VALIDBLOCKLIST;

else

sendB[0] = (byte)Header.NOTVALIDBLOCKLIST;

blockFile.Close();

BinSender bsend = new BinSender(storage);

bsend.ExceptionReceived += ExceptionReceived;

bsend.SendValid(((IPEndPoint)handler.RemoteEndPoint).Address, sendB);

}

catch (Exception e)

{

ExceptionReceived(this, e);

}

break;

}

case Header.VALIDBLOCKLIST:

{

storage.Verified++;

if (storage.Verified > (storage.userIPList.Count / 2))

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

storage.blockList = bs.GetBlockList();

}

break;

}

case Header.ADDTRANSACTION:

{

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += ExceptionReceived;

Transaction tr;

ms.Seek(1, SeekOrigin.Begin);

tr = bs.DeserializeTransaction(ms);

TransactionVerifier tv = new TransactionVerifier();

storage.blockList.Last().trans\_list.Add(tr);

storage.blockList.Last().hash = tv.PackBlock(storage.blockList.Last());

if (storage.blockList.Last().trans\_list.Count == 10)

{

storage.blockList.Last().hash = tv.PackBlock(storage.blockList.Last());

Block block = new Block();

block.hash\_before = storage.blockList.Last().hash;

block.trans\_list = new List<Transaction>();

uint[] rec = tv.SearchSender(storage, tr.in\_list.ElementAt(0).hash, tr.in\_list.ElementAt(0).n);

storage.blockList.Add(block);

tv.BonusTonkoin(storage, rec);

block.hash = tv.PackBlock(block);

}

bs.SaveBlockList(storage.blockList);

storage.newTransaction = null;

this.TransactionReceived(this, tr);

break;

}

}

handler.Shutdown(SocketShutdown.Both);

ms.Close();

}

}

}

}

Класс TransactionVerifier

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using tonkoin.Serializers;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

class TransactionVerifier

{

public TransactionVerifier() { }

public int TransactionUsed(Transaction tr, List<Block> blockList)

{

uint[] outHash = tr.hash;

int res = 0;

Sha256Digest sha = new Sha256Digest();

bool found = false;

foreach (Block block in blockList)

{

foreach (Transaction trans in block.trans\_list)

{

if (sha.IsEqualHash(trans.hash, tr.hash)) //если транзакция в этом блоке, с него надо начинать проверку

{

found = true;

}

}

if (found)

{

foreach (Transaction trans in block.trans\_list)

{

foreach (In inElement in trans.in\_list)

{

if (sha.IsEqualHash(outHash, inElement.hash))

{

if (inElement.n == 0)

res += 1; // использовался 1 выход

if (inElement.n == 1)

res += 2; // использовался 2 выход

}

}

}

}

}

return res;

}

public Transaction IsTransactionValid(List<Block> blockList, string from, string dest, string password, int value)

{

int ttl = 0;

Sha256Digest sha = new Sha256Digest();

uint[] from\_hash = sha.hash(Encoding.UTF8.GetBytes(from));

sha = new Sha256Digest();

uint[] dest\_hash = sha.hash(Encoding.UTF8.GetBytes(dest));

int used = 0;

List<In> listIn = new List<In>();

if (blockList != null)

{

foreach (Block block in blockList)

{

foreach (Transaction tr in block.trans\_list)

{

if ((used = TransactionUsed(tr, blockList)) != 3) //не все выходы использованы

{

if (used != 1) // не использован первый выход

if (sha.IsEqualHash(tr.out\_list.ElementAt(0).dest\_hash, from\_hash))

{

In in\_tr = new In(tr.hash, 0, password);

listIn.Add(in\_tr);

ttl += tr.out\_list.ElementAt(0).value;

if (ttl >= value)

{

break;

}

}

if (used != 2) // не использован второй выход

if (tr.out\_amount == 2)

if (sha.IsEqualHash(tr.out\_list.ElementAt(1).dest\_hash, from\_hash))

{

In in\_tr = new In(tr.hash, 1, password);

listIn.Add(in\_tr);

ttl += tr.out\_list.ElementAt(1).value;

if (ttl >= value)

{

break;

}

}

}

}

}

}

if ((listIn.Count == 0) || (ttl < value))

{

return null;

}

if (ttl < value)

{

return null;

}

List<Out> listOut = new List<Out>();

Out out\_tr = new Out(dest\_hash, value);

listOut.Add(out\_tr);

if (ttl > value)

{

Out out\_tr2 = new Out(from\_hash, ttl - value);

listOut.Add(out\_tr2);

}

Transaction newTr = new Transaction(listIn, listOut);

return newTr;

}

public bool IsTransactionValid(Storage st, Transaction trans)

{

Sha256Digest sha = new Sha256Digest();

int used;

foreach (Block block in st.blockList)

{

foreach (Transaction tr in block.trans\_list)

{

foreach (In inTr in trans.in\_list)

{

if (sha.IsEqualHash(inTr.hash, tr.hash))

{

used = TransactionUsed(trans, st.blockList);

if (used == 3)

{

return false;

}

if (inTr.n == 0)

{

if (used == 1)

{

return false;

}

}

if (inTr.n == 1)

{

if (used == 2)

{

return false;

}

}

}

}

}

}

return true;

}

public int ShowBallance(Storage st)

{

uint[] usHash = st.curentUser.hash;

int amount = 0;

int used;

Sha256Digest sha = new Sha256Digest();

foreach (Block bl in st.blockList)

{

foreach (Transaction trans in bl.trans\_list)

{

used = TransactionUsed(trans, st.blockList);

if (used != 3)

{

if (sha.IsEqualHash(trans.out\_list.ElementAt(0).dest\_hash, usHash))

{

if (used != 1)

{

amount += trans.out\_list.ElementAt(0).value;

}

}

if (trans.out\_list.Count > 1)

{

if (sha.IsEqualHash(trans.out\_list.ElementAt(1).dest\_hash, usHash))

if (used != 2)

{

amount += trans.out\_list.ElementAt(1).value;

}

}

}

}

}

return amount;

}

public uint[] PackBlock(Block block)

{

uint[] hash = block.trans\_list.ElementAt(0).hash;

for(int i = 1; i < block.trans\_list.Count; i++)

{

uint[] addhash = block.trans\_list.ElementAt(i).hash;

for (int j = 0; j < 8; j++)

hash[j] = hash[j] ^ addhash[j];

}

return hash;

}

private uint[] FindParent(Storage st, uint[] rec)

{

Sha256Digest sha = new Sha256Digest();

User user = null;

foreach (User us in st.userList)

{

if (sha.IsEqualHash(us.hash, rec))

user = us;

}

foreach (User us in st.userList)

{

if (sha.IsEqualHash(us.hash, user.parent\_hash))

return us.hash;

}

return null;

}

public void BonusTonkoin(Storage st, uint[] rec)

{

Sha256Digest sha = new Sha256Digest();

Out out1 = new Out(rec, 20);

uint[] ui = new uint[1];

ui[0] = 0;

In in1 = new In(ui, 0, "");

List<In> inList = new List<In>();

inList.Add(in1);

List<Out> outList = new List<Out>();

outList.Add(out1);

Transaction trans = new Transaction(inList, outList);

st.blockList.Last().trans\_list.Add(trans);

uint[] parentHash = FindParent(st, rec);

while(parentHash != null)

{

out1 = new Out(parentHash, 1);

in1 = new In(ui, 0, "");

inList = new List<In>();

inList.Add(in1);

outList = new List<Out>();

outList.Add(out1);

trans = new Transaction(inList, outList);

st.blockList.Last().trans\_list.Add(trans);

parentHash = FindParent(st, parentHash);

}

}

public uint[] SearchSender(Storage st, uint[] hash, int x)

{

Sha256Digest sha = new Sha256Digest();

foreach (Block bl in st.blockList)

{

foreach (Transaction trans in bl.trans\_list)

{

if (sha.IsEqualHash(hash, trans.hash))

return trans.out\_list.ElementAt(x).dest\_hash;

}

}

return hash;

}

}

}

Класс UserIP

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

namespace tonkoin

{

[Serializable]

public class UserIP

{

public string ID { get; set; }

public IPAddress IP { get; set; }

public UserIP()

{ }

public UserIP(string ID, IPAddress IP)

{

this.ID = ID;

this.IP = IP;

}

}

}

Класс UserIPSerializer

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Net;

namespace tonkoin

{

class UserIPSerializer

{

public UserIPSerializer() { }

public byte[] Serialize(List<UserIP> uL)

{

String retS = "";

foreach (UserIP uI in uL)

{

retS += uI.ID + "~" + uI.IP.ToString() + "~";

}

return Encoding.UTF8.GetBytes(retS);

}

public List<UserIP> Deserialize(byte[] b)

{

string[] s = Encoding.UTF8.GetString(b).Split(new char[] { '~' });

List<UserIP> userIpL = new List<UserIP>();

for (int i = 0; i < s.Length - 1; i += 2)

{

UserIP usIp = new UserIP();

usIp.ID = s[i];

usIp.IP = IPAddress.Parse(s[i + 1]);

userIpL.Add(usIp);

}

return userIpL;

}

}

}

Класс RegistrationForm

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using tonkoin.Serializers;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

public partial class RegistrationForm : Form

{

public RegistrationForm()

{

InitializeComponent();

}

private void OnExceptionReceived(object sender, Exception e)

{

MessageBox.Show(e.ToString());

}

private void backBT\_Click(object sender, EventArgs e)

{

Close();

}

private void registerBT\_Click(object sender, EventArgs e)

{

Tonkoin owner = (Tonkoin)this.Owner;

BinSerializer bs = new BinSerializer();

bs.ExceptionReceived += OnExceptionReceived;

String login, password, parent;

login = loginTB.Text;

if (login == "")

{

MessageBox.Show("Please, type your login.");

return ;

}

if (passwordTB.Text == "")

{

MessageBox.Show("Please, type your password.");

return;

}

if (passwordTB.Text != passwordAgainTB.Text)

{

MessageBox.Show("Password again does not match.");

return;

}

password = passwordAgainTB.Text;

parent = parentTB.Text;

if (parent == "")

{

MessageBox.Show("Please, login of user, wh invited you.");

return;

}

owner.storage.curentUser = new User(login, parent, password);

if (owner.storage.curentUser.IsUserExists(owner.storage.userList, owner.storage.curentUser.ID))

{

MessageBox.Show("User " + owner.storage.curentUser.ID + " already exists.");

return;

}

if (!(owner.storage.curentUser.IsUserExists(owner.storage.userList, parent)))

{

MessageBox.Show("User " + parent + " does not exists.");

return;

}

owner.storage.userList.Add(owner.storage.curentUser);

bs.SaveUserList(owner.storage.userList);

if (owner.storage.userIPList != null)

{

BinSender binSender = new BinSender(owner.storage);

binSender.ExceptionReceived += OnExceptionReceived;

binSender.SendUserList();

}

MessageBox.Show("Success.");

}

}

}

Класс TransactionForm

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using tonkoin.Serializers;

using TonkoinLibrary.Transactions;

namespace tonkoin

{

public partial class TransactionForm : Form

{

public TransactionForm()

{

InitializeComponent();

}

private void backBT\_Click(object sender, EventArgs e)

{

this.Close();

}

private void OnExceptionReceived(object sender, Exception e)

{

MessageBox.Show(e.ToString());

}

private void sendTransBT\_Click(object sender, EventArgs e)

{

if (sendTransBT.Text == "Send")

{

String login, password;

byte value;

Tonkoin owner = (Tonkoin)this.Owner;

login = receiverTB.Text;

if (login == "")

{

MessageBox.Show("Please, type receiver login.");

return;

}

if (valueTB.Text == "")

{

MessageBox.Show("Please, type value to send.");

return;

}

try

{

value = Convert.ToByte(valueTB.Text);

}

catch

{

MessageBox.Show("Please, enter valid value.");

return;

}

password = passwordTB.Text;

if (password == "")

{

MessageBox.Show("Please, type receiver login.");

return;

}

if (!(owner.storage.curentUser.IsUserExists(owner.storage.userList, login)))

{

MessageBox.Show("User " + login + " not found.");

return;

}

TransactionVerifier tv = new TransactionVerifier();

Transaction newTr;

if ((newTr = tv.IsTransactionValid(owner.storage.blockList, owner.storage.curentUser.ID, login, password, value)) != null)

{

BinSender bs = new BinSender(owner.storage);

bs.ExceptionReceived += OnExceptionReceived;

bs.SendTransaction(owner.storage.userIPList, owner.storage.curentUserIP, newTr);

}

}

}

}

}

Проект TonkoinLibrary

Класс Transaction

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TonkoinLibrary.Transactions

{

[Serializable]

public class Transaction

{

public uint[] hash;

public int in\_amount; //Количество предыдущих транзакций, из которых деньги переводятся на новые адреса. Одна или более.

public int out\_amount; //Количество адресов, на которые переводятся деньги. Один или более.

public int size;

public List<In> in\_list;

public List<Out> out\_list;

public Transaction()

{ }

public Transaction(List<In> in\_list, List<Out> out\_list)

{

this.in\_list = in\_list;

this.out\_list = out\_list;

this.in\_amount = in\_list.Count;

this.out\_amount = out\_list.Count;

this.size = sizeof(uint) \* 8 + sizeof(int) \* 3; // неправильно!

Sha256Digest sha = new Sha256Digest();

String s;

DateTime dt = DateTime.Now;

s = dt.Millisecond.ToString() + in\_list.ElementAt(0).hash[0].ToString() + out\_list.ElementAt(0).dest\_hash[7].ToString();

this.hash = sha.hash(Encoding.UTF8.GetBytes(s));

}

}

}

Класс User

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TonkoinLibrary.Transactions

{

[Serializable]

public class User

{

public uint[] hash;

public String ID;

public uint[] parent\_hash;

public uint[] password\_hash;

public User(String ID, String parent, String password)

{

Byte[] s;

Sha256Digest sha = new Sha256Digest();

s = Encoding.UTF8.GetBytes(password);

this.password\_hash = sha.hash(s);

s = Encoding.UTF8.GetBytes(ID);

sha = new Sha256Digest();

this.hash = sha.hash(s);

this.ID = ID;

sha = new Sha256Digest();

s = Encoding.UTF8.GetBytes(parent);

this.parent\_hash = sha.hash(s);

sha = new Sha256Digest();

}

public bool IsUserExists(List<User> userList, String login)

{

foreach (User user in userList)

{

if (user.ID == login)

return true;

}

return false;

}

}

}

Класс Out

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TonkoinLibrary.Transactions

{

[Serializable]

public class Out

{

public int value; // Содержит количество денег, которые будут переведены по новому адресу.

public uint[] dest\_hash; // Хэш получателя

public Out(uint[] dest\_hash, int value)

{

this.dest\_hash = dest\_hash;

this.value = value;

}

}

}

Класс In

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TonkoinLibrary.Transactions

{

[Serializable]

public class In

{

public uint[] hash; //хэш предыдущей транзакции

public int n; //номер выхода, с которого нужно брать деньги

public uint[] sig;

public In(uint[] hash, int n, String s)

{

this.hash = hash;

this.n = n;

Sha256Digest sha = new Sha256Digest();

this.sig = sha.hash(Encoding.UTF8.GetBytes(s));

}

}

}

Класс Block

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TonkoinLibrary.Transactions

{

[Serializable]

public class Block

{

public uint[] hash;

public uint[] hash\_before;

public List<Transaction> trans\_list; //список транзакций

int repeat; //для формирования правильного хэша

public int trans\_amount;

public int size;

public Block() { }

}

}

Проект Alphach (сервер)

Класс Program

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Net;

using System.Net.Sockets;

using System.Runtime.Serialization.Formatters.Binary;

using System.IO;

namespace Alphach

{

class Program

{

static List<UserIP> userIPList = new List<UserIP>();

static void Main(string[] args)

{

IPEndPoint ipEndPoint = new IPEndPoint(IPAddress.Parse("192.168.43.89"), 11000);

Socket sListener = new Socket(ipEndPoint.AddressFamily, SocketType.Stream, ProtocolType.Tcp);

sListener.Bind(ipEndPoint);

sListener.Listen(10);

Console.WriteLine("Server working {0}", ipEndPoint);

while (true)

{

string data;

byte[] bytes = new byte[2000];

Socket handler = sListener.Accept();

Console.WriteLine("Connection from " + handler.RemoteEndPoint.ToString());

int bytesRec = handler.Receive(bytes);

{

if ((bytesRec == 1) && (bytes[0] == 0))

{

int i = 0;

foreach (UserIP usIP in userIPList)

{

if (usIP.IP.ToString() == (((IPEndPoint)handler.RemoteEndPoint).Address).ToString())

userIPList.RemoveAt(i);

break;

i++;

}

continue;

}

}

Deserializer ds = new Deserializer();

List<UserIP> getList = new List<UserIP>();

getList = ds.Deserialize(bytes);

getList.ElementAt(0).IP = ((IPEndPoint)handler.RemoteEndPoint).Address;

bool found = false;

foreach(UserIP uI in getList)

{

foreach(UserIP uIp in userIPList)

{

if (uI.ID == uIp.ID)

found = true;

if (uI.IP.ToString() == uIp.IP.ToString())

{

uIp.ID = uI.ID;

found = true;

}

}

if (!found)

userIPList.Add(uI);

found = false;

}

Serializer s = new Serializer();

bytes = s.Serialize(userIPList);

handler.Send(bytes);

}

}

}

}