МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Клиент-серверное приложение для распределенной расшифровки сверток паролей

Студент 03.06.2020 Кривошеев Сергей Валерьевич
(подпись, дата) (ФИО полностью)

Руководитель 02.06.2020 А.Н. Токарев
(подпись, дата)

Нормоконтролёр А.В. Кучин
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 03.06.2020г. № 12)

Заведующий кафедрой М.А. Митрохин
(подпись) (фамилия, инициалы)

Работа защищена с отметкой хорошо (протокол заседания ГЭК от 23.06.2020 № 19)

Секретарь ГЭК Е.И. Гурин
(подпись) (фамилия, инициалы)

Пенза 2020

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет вычислительной техники

Кафедра «Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«30» апреля 2020г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

1. Студент Кривошеев Сергей Валерьевич гр. 16ВВ1 факультета вычислительной техники.

2.Тема работы «Клиент-серверное приложение для распределенной расшифровки сверток паролей»

Тема утверждена приказом ПГУ №0335а/о от "30" апреля 2020г.

3. Руководитель бакалаврской работы кандидат технических наук, доцент кафедры ВТ Токарев А.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Назначение разработки – клиент-серверное приложение, представляющее из себя параллельную программу для кластера облачного типа, которое позволяет расшифровывать свертки паролей. Необходимо предусмотреть возможность одновременного подключения нескольких клиентов, разделение задачи по подзадачи, раздача подзадач клиентам и получение ответа. Серверная и клиентская часть должна иметь графический интерфейс для комфортной работы.

5. Перечень подлежащих разработке вопросов
Требуется произвести анализ предметной области, выбрать архитектуру приложения, определить языки программирования для клиентского и серверного приложения, выбрать способ взаимодействия клиента и сервера, разработать протокол взаимодействия, произвести тестирование в одиночном и распределённом режимах

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области	10	30.04.2020 – 04.05.2020	
2	Выбор архитектуры приложения, способа взаимодействия	10	04.05.2020 – 07.05.2020	
3	Выбор языка программирования и среды разработки	10	07.05.2020 – 11.05.2020	
4	Разработка протокола взаимодействия	10	11.05.2020 – 15.05.2020	
5	Разработка серверной части	20	15.05.2020 – 22.05.2020	
6	Разработка клиентской части	10	22.05.2020 – 25.05.2020	
7	Тестирование и отладка	10	25.05.2020 – 28.05.2020	
8	Оформление выпускной работы	20	28.05.2020 – 03.06.2020	

Дата выдачи задания "30" апреля 2020г.

Руководитель бакалаврской работы 30.04.2020 Токарев А.Н.
(подпись, дата)

Задание к исполнению принял студент 30.04.2020 Кривошеев С.В.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ Фионова Л.Р.
(подпись, дата)

Реферат

Выпускная квалификационная работа бакалавра 120 листов, 55 рисунков, 3 таблицы, 6 источников, 6 приложения.

КЛИЕНТ-СЕРВЕР, ПРИЛОЖЕНИЕ, СЕРВЕР, КЛИЕНТ, ПРОГРАММА, HASHCAT, ОБЕСПЕЧЕНИЕ, ПОДЗАДАЧА, ЗАДАЧА, РАСШИФРОВКА, ВОССТАНОВЛЕНИЕ ПАРОЛЯ, ПАРОЛЬ, РАСПРЕДЕЛЕННАЯ СИСТЕМА

Цель работы — разработать клиент-серверное приложение, осуществляющее распределение задачи на несколько подзадач, отправку зашифрованного файла и подзадачи клиентам для расшифровки.

Объектом разработки является клиент-серверное приложение, состоящее из программы с графическим интерфейсом и открытого программного обеспечения hashcat. Сервер работает в асинхронном режиме, тем самым осуществляя подключение нескольких клиентов и раздачу им подзадач. Клиент принимает файл и подзадачи от сервера с последующим запуском программы hashcat со специальным опциями.

В результате проведенной работы реализован проект, удовлетворяющий поставленной задаче, а именно: создано клиент-серверное приложение, распределяющее задачу на подзадачи и осуществляющее распределение их по подключенным клиентам с последующим запуском программного обеспечения hashcat для расшифровки.

Данное приложение рекомендуется использовать для восстановления забытых паролей. Приложение может быть внедрено на предприятиях, которые осуществляют пентест (испытание на проникновение) и анализ уязвимостей. Так же приложение может использоваться простыми пользователями для восстановления забытых паролей.

Содержание

Введение **6**

1. Обзор предметной области **8**

1.1 Постановка задачи 8

1.2 Анализ требований для реализации сервера 9

1.3 Анализ требований для реализации клиента 10

1.4 Выбор языка программирования 11

1.5 Win Forms API для создания графического интерфейса 12

1.6 Выбор системы распределенных вычислений 13

2. Выбор протокола передачи данных **15**

2.1 Протокол UDP 15

2.2 Протокол TCP 15

3 Выбор программы для распределенного расчета **18**

4 Разработка стратегии распределения подзадач **23**

5 Реализация клиент-серверного приложения **24**

5.1 Описание работы клиент-серверного приложения 24

5.1.1 Описание работы клиента 24

5.1.2 Описание работы сервера 25

5.2 Разработка серверной части приложения 26

5.3 Разработка клиентской части приложения 32

6 Описание приложения **37**

6.1 Руководство пользователя сервера 37

6.2 Руководство пользователя клиента 46

6.3 Спецификация сервера 47

6.4 Спецификация клиента 49

6.5 Спецификация библиотеки классов 50

7 Тестирование в одиночном и распределенном режимах **51**

7.1 Результат тестирования в одиночном режиме 51

7.2 Результат тестирования в распределенном режиме 52
8 Безопасность жизнедеятельности 55
8.1 Общие требования безопасности 55
8.2 Требования безопасности перед началом работы 56
Заключение 57
Список использованных источников 59
Приложение А Листинги программы сервера 60
Приложение А.1 Файл «Form1.cs» 60
Приложение А.2 Файл «DataTask.cs» 71
Приложение А.3 Файл «AboutBox1.cs» 72
Приложение А.4 Файл «Client.cs» 73
Приложение А.5 Файл «addtask.cs» 73
Приложение А.6 Файл «GetIP.cs» 77
Приложение А.7 Файл «myQueue.cs» 77
Приложение А.8 Файл «Outfile.cs» 78
Приложение А.9 Файл «Queuetask.cs» 79
Приложение А.9 Файл «Queuezadach.cs» 80
Приложение А.9 Файл «raspredtask.cs» 82
Приложение А.10 Файл «spisoktask.cs» 82
Приложение В Листинги программы клиента 83
Приложение В.1 Файл «Form1.cs» 83
Приложение В.2 Файл «SocketPacket.cs» 88
Приложение В.3 Файл «WorkHash.cs» 88
Приложение С Листинги библиотеки классов 90
Приложение С.1 Файл «DataInClient.cs» 90
Приложение С.2 Файл «DataInServer.cs» 90
Приложение D Результаты работы программы 91
Приложение D.1 Вид начально экрана клиента 91
Приложение D.2 Вид свернутого в трей клиента 91
Приложение D.3 Ошибка при подключении к серверу 91

Приложение D.4 Сообщение об отключении сервера..... 92
Приложение D.5 Вид начально экрана сервера 92
Приложение D.6 Вид экрана с добавленной задачей 93
Приложение D.7 Окно выбора файла handshake..... 93
Приложение D.8 Вид экрана с сообщением об ошибке 1..... 94
Приложение D.9 Вид экрана с сообщением об ошибке 2..... 94
Приложение D.10 Вид экрана с сообщением об ошибке 3..... 95
Приложение D.11 Контекстное меню 95
Приложение D.12 Вид экрана при расшифровке..... 96
Приложение D.13 Вид экрана при удачном поиске пароля 96
Приложение D.14 Вид экрана работы сервера..... 97
Приложение D.15 Вид экрана при удалении задачи 97
Приложение D.16 Вид таблицы подзадач 98
Приложение D.17 Выгруженная таблица в Excel..... 99
Приложение E UML-диаграммы приложения..... 103
Приложение E.1 – UML-диаграмма классов сервера..... 103
Приложение E.2 – UML-диаграмма классов клиента 104
Приложение E.3 – UML-диаграмма вариантов использования сервера 105
Приложение E.4 – UML-диаграмма вариантов использования клиента..... 105
Приложение E.5 – UML-диаграмма деятельности сервера 106
Приложение E.6 – UML-диаграмма деятельности клиента..... 107
Приложение E.7 – UML-диаграмма последовательности 108
Приложение E.8 – UML-диаграмма развертывания..... 108
Приложение F Презентация..... 109

Введение

В настоящее время во многих областях деятельности человека существует большое количество как научно-фундаментальных, так и прикладных задач, решение которых возможно только с использованием высокопроизводительных вычислений.

Для решения этих задач возникает потребность в значительно больших вычислительных ресурсах, чем может предоставить обычный персональный компьютер. Для их решения необходимо большее быстродействия, либо обрабатывать и хранить большие объемы информации.

Для организации высокопроизводительных систем исторически используются суперкомпьютеры - особым образом сконструированные вычислительные комплексы, построенные на специализированных комплектующих и обеспечивающие максимальную производительность. Данные системы имеют большую стоимость как в приобретении, так и в обслуживании.

Однако в некоторых случаях для организации распределённой вычислительной системы подобные решение считаются дорогими и избыточно производительными.

Кластерные системы возникли как более дешевое решение проблемы недостатка вычислительных ресурсов, и основываются на использовании в своей архитектуре широко распространенных и относительно дешевых технологий, аппаратных и программных средств, таких как PC, Ethernet, Linux и т.д. Использование массовых технологий в кластерных системах стало возможным благодаря значительному прогрессу в развитии компонентов обычных вычислительных систем, таких как центральные процессоры, операционные системы, коммуникационные среды.

Преимущества такой системы: возможность оптимального размещения задачи по вычислительным узлам разной архитектуры и различной мощности,

возможность динамического подключения и отключения произвольного количества вычислительных узлов.

Раньше для того, чтобы программировать на распределенной вычислительной системе требовались глубокие знания математического аппарата и программирования. В настоящее время имеется тенденция к организации доступа к подобным системам на уровне пользователя - посредством клиент-серверного приложения.

По этим причинам,

Объект исследования: клиент-серверный объект распределенных вычислений кластерного типа.

Предмет исследования: организация процесса восстановления пароля в распределенной вычислительной среде и управления посредством графического интерфейса на серверном приложении.

Практическая ценность: повышение производительности при восстановлении паролей. Создание программного обеспечения для реализации на реальной вычислительной системе.

1. Обзор предметной области

1.1 Постановка задачи

Требуется разработать клиент-серверное приложение, распределяющие задачи по подключенным клиентам для поиска пароля по его хэш-значению:

1. Серверное приложение должно предоставлять возможность подключения нескольких клиентов и отправку каждому из клиентов подзадачи.

2. Серверное приложение должно предоставлять возможность пользователю настраивать, добавлять, просматривать, запускать расшифровку и удалять задачи.

3. Каждая задача при добавлении должна добавляться в список задач и разбиваться на подзадачи.

4. Каждую задачу можно открыть и посмотреть ее подзадачи и информацию о добавленной задаче.

5. При удалении задачи, серверное приложение должно:

1) отправить команду об остановке расшифровки определенному клиенту, задача которого подлежит удалению;
2) вернуть данного клиента в очередь клиентов;
3) выполнить очистку очереди подзадач;
4) удалить выбранную задачу из очереди задач, если она присутствует в ней;
5) выполнить очистку подзадач удаляемой задачи из списка подзадач;
6) выполнить удаление задачи из списка задач.

6. При нахождении пароля, серверное приложение должно сделать аналогичные действия, что и при удалении задачи, кроме удаления задачи из списка задач и очистки подзадач.

7. При отключении клиента, удалять его из очереди клиентов, если клиент работал над подзадачей, возвращать данную подзадачу в очередь подзадач.

8. Все действия на серверном приложении фиксируются в консоли.

9. Клиентская часть должна давать возможность подключаться к удаленной машине.

10. При получении задачи от сервера, клиентское приложение должно запустить программу hashcat и ожидать ее завершение.

11. При получении команды об остановке расшифровки, клиентское приложение должно остановить работу программы hashcat.

12. После работы программы hashcat, клиентское приложение должно отослать серверу ответы в виде пароля, либо сообщение, что пароль не найден.

13. Приложения должны работать в ОС Microsoft Windows 7 и выше.

Задача - совокупность информации о нахождении файла с хэш-значением, именем файла, длины пароля, набор символов, названии задачи.

Подзадача - некая часть от задачи, имеющая опцию для запуска hashcat.

Реализовав данный функционал, можно решить следующие задачи:

1. Восстановление забытых паролей пользователя при нерациональном или невозможном изменения его административными методами;
2. Восстановление зашифрованных данных;
3. Предоставление разработчикам криптографических алгоритмов информацию об усовершенствовании алгоритма;
4. Расшифровка зашифрованных вирусами файлов.

1.2 Анализ требований для реализации сервера

Сервер – это компьютерная программа или устройство выполняющие услуги для других программ или устройств, которые называются клиентами. При выборе технологий для реализации сервера нужно учитывать их влияние на быстродействие, отказоустойчивость и масштабируемость.

Сервер должен выполнять различные функции: хранение данных, их распределение между клиентами, выполнение с ними различных вычислений.

Один сервер может обслуживать несколько клиентов, а один клиент может использовать несколько серверов. Часто сервер воспринимается, как удаленный

компьютер, обслуживающий запросы. На самом деле клиентское и серверное приложение могут находится на одной машине.

Существует множество типов серверов, несколько из них приведены ниже:

1. Сервер приложений – это программа на компьютере в распределенной сети, которая обеспечивает бизнес-логику для прикладной программы.

2. Прокси-сервер – это элемент сетевой инфраструктуры, который выполняет роль посредника между клиентским компьютером, находящимся во внутренней сети и другим сервером, который находится во внешней сети или внутренней.

3. Почтовый сервер – сервер, расположенный в Интернете или интрасети, принимающий или передающий сообщения электронной почты по протоколам SMTP, POP, IMAP.

4. Виртуальный сервер – виртуальный объект, созданный в специализированном программном обеспечении и имеющий такой же набор характеристик, как и у обычного физического сервера, который может быть развернут на любом физическом сервере.

5. Блейд-сервер – это серверное шасси с множеством тонких, модульных электронных плат, известных как blade-серверы. Каждый blade-сервер является отдельным сервером, часто предназначенным для одного приложения.

6. Файловый сервер – это компьютер, отвечающий за централизованное хранение и управление данными, чтобы другие компьютеры в одной сети могли получить к ним доступ.

7. Сервер политики – это компонент безопасности сети на основе политик, которая предоставляет услуги авторизации и облегчает отслеживание и управление доступом к данным.

1.3 Анализ требований для реализации клиента

Клиентское приложение – работающая на компьютере пользователя и обеспечивающая взаимодействие системы с пользователем. Инструменты, его

реализующие, должны позволять создать интерфейс, обеспечивающий удобное использование и взаимодействия с ним.

Современное приложение должно иметь пользовательский интерфейс, выполненный в современном дизайнерском стиле с использованием спокойных цветов. Клиентская часть должна иметь грамотное расположение элементов, что позволит пользователю легко использовать и ориентироваться в интерфейсе. Следует принимать во внимание группы пользователей с различными ограничениями, связанные с заболеваниями. Необходимо продумать и реализовать инструменты, с помощью которых они бы смогли полноценно использовать все возможности приложения наравне с другими.

Интегрированная среда разработки должна поддерживать средства отображения и редактирования графической части, написанной на выбранном языке программирования. Это обеспечит не только удобство при разработке и дальнейшей поддержки приложения, но и позволит в дальнейшем вносить в него изменения человеку без навыков программирования.

1.4 Выбор языка программирования

Учитывая требования, которые описаны в предыдущих пунктах, для реализации сервера и клиента выбор пал на язык программирования C#. C# является объектно-ориентированным языком, но поддерживает также и компонентно-ориентированное программирование. Можно выделить основные черты языка:

1. Поддержка нескольких языков
2. Кроссплатформенность.
3. Мощная библиотека классов.
4. Разнообразие технологий

Так же язык C# предоставляется функции для обеспечения надежность и устойчивость приложений. Сборка мусора автоматически освобождает память. Обработка исключений предоставляет структурированный и расширяемый подход к обнаружению ошибок и их восстановлению. Типобезопасная структура

языка делает невозможным чтение из неинициализированных переменных, индексацию массивов за пределами их границ или выполнение непроверенных приведений типов.

Помимо основных принципов ООП (инкапсуляция, наследование, полиморфизм), C# поддерживает ряд особенных языковых конструкций, облегчающих разработку компонентов.

1. Делегаты – представляет собой объект, который может ссылаться на метод. Другими словами, делегат позволяет вызывать метод, на который он ссылается.

2. Свойства – обеспечивают простой доступ к полям классов и структур, узнать их значение или выполнить их установку.

3. Атрибуты – позволяют программам указывать дополнительные описательные данные о типах, членах и других сущностях.

Благодаря этому он отлично подходит для создания и применения программных компонентов.

1.5 Win Forms API для создания графического интерфейса

Приложения с графическим интерфейсом могут быть созданы на языке C# с использованием Windows Forms. Windows Forms – это способ конструирования приложений на базе классов .NET Framework class library. В них используется собственная модель программирования, которая совершеннее, чем основанные на Win32 API или MFC. Win Forms приложения выполняются в управляемой среде .NET Common Language Runtime (CLR).

Главная выгода от написания Windows-приложений с использованием – Windows Forms – то, что он гомогенизирует (строит однородную структуру) программную модель и устраняет многие ошибки и противоречия, возникающие при работе с Windows API.

NET Framework class library содержит намного больший набор классов для реализации графической части любой сложности. Приложения, написанные с

использованием Windows Forms, содержат меньше кода, чем аналогичные, но использующие Windows API или MFC.

Другая выгода от Windows Forms – это наличие идентичного API для набора разных языка программирования. Все приложения, которые используют Windows Forms, используют один API из .NET Framework class library. Знание одного API позволяет программисту писать приложения фактически на любом языке, поддерживаемый .NET Framework.

1.6 Выбор системы распределенных вычислений

Проанализировав существующие системы распределенных вычислений, можно выбрать систему, которое даст наибольшую отдачу в настоящем и ближайшем будущем в рамках имеющейся инфраструктуры вычислительной техники.

Учитывая такие факторы как:

- гибкость в выборе языков программирования и средств вычислений,
- ориентированность на существующую инфраструктуру сетей вычислительной техники,
- минимальная стоимость программного и аппаратного обеспечения,
- быстрое и простое развертывание и настройка системы,
- высокая степень приспособляемости к изменяющимся условиям работы,
- высокая надежность системы и невосприимчивость к отказам отдельных узлов и линий связи,

Из всего вышеописанного в качестве системы распределённых вычислений была выбрана кластерная система. В нашем случае — это совокупность компьютеров, объединенных в рамках некоторой сети для решения одной задачи, которая для пользователя представляется в качестве единого ресурса.

Вычислительные машины (системы) в кластере взаимодействуют в соответствии с одним из транспортных протоколов:

1. Протокол TCP (Transmission Control Protocol), оперирует потоками байтов, гарантируя надежность доставки сообщения.

2. Проткол UDP (User Datagram Protocol) пытается посылать пакеты данных без гарантии их доставки.

Неотъемлемая часть кластера - специализированное программное обеспечение, организующее работу кластерной системы. Такое программное обеспечение должно быть установлено на каждый узел кластера. Оно реализует механизм передачи сообщений через стандартные сетевые протоколы. Именно благодаря этому вычислительные машины, объединенных сетью, превращается в кластерную вычислительную систему.

Есть несколько подходов к созданию кластерной системы, в данной работе используется подход для построения небольших кластерных систем, например, на базе небольших локальных сетей организаций или их подразделений. В таком кластере каждая вычислительная машина продолжает работать как самостоятельная единица, одновременно выполняя функции узла кластерной системы.

2. Выбор протокола передачи данных

2.1 Протокол UDP

Протокол UDP является дейтаграммным протоколом, реализующую ненадёжную передачу данных, который не гарантирует доставку сообщений. При работе на хосте-отправителе данные от приложений поступают протоколу UDP через порт в виде сообщений.

Каждая дейтаграмма переносит отдельное пользовательское сообщение. Сообщения могут иметь разную длину, не превышающую, длину поля данных протокола IP, которое, в свою очередь, ограничено размером кадра технологии нижнего уровня. Поэтому если буфер UDP переполняется, то сообщение приложения отбрасывается.

Заголовок UDP состоит из четырех двухбайтных полей:

1. номер UDP-порта отправителя;
2. номер UDP-порта получателя;
3. контрольная сумма;
4. длина дейтаграммы.

Простота заголовка дает делает реализацию UDP протокола не сложной. Его функции сводятся к простой передаче данных между прикладным и сетевым уровнями, а также примитивному контролю искажений в передаваемых данных. При контроле искажений протокол UDP только диагностирует, но не исправляет ошибку. Если контрольная сумма показывает, что в поле данных UDP-дейтаграммы произошла ошибка, то протокол UDP просто отбрасывает поврежденную дейтаграмму.

2.2 Протокол TCP

При работе на хосте-отправителе протокол TCP рассматривает информацию, поступающую к нему от прикладных процессов, как неструктурированный поток байтов.

Поступающие данные буферизуются средствами TCP. Для передачи на сетевой уровень из буфера «вырезается» некоторая непрерывная часть данных, которая называется сегментом и снабжается заголовком.

В отличие от протокола UDP, создающего дейтаграммы на основе логически обособленных единиц данных, протокол TCP делит поток данных на сегменты без учета их смысла или внутренней структуры.

Логические соединения — основа надежности TCP

Основным отличием TCP от UDP является то, что на протокол TCP возложена дополнительная задача — обеспечить надежную доставку сообщений, используя в качестве основы ненадежный дейтаграммный протокол IP.

Для решения этой задачи протокол TCP использует метод продвижения данных с установлением логического соединения. Как отмечалось, логическое соединение дает возможность участникам обмена следить за тем, чтобы данные не были потеряны, искажены или продублированы, а также чтобы они пришли к получателю в том порядке, в котором были отправлены.

Протокол TCP устанавливает логические соединения между прикладными процессами, причем в каждом соединении участвуют только два процесса. TCP-соединение является дуплексным, то есть каждый из участников этого соединения может одновременно получать и отправлять данные.

В первую очередь, серверное приложение должно создать соединение с известным номером порта, а затем перейти в режим прослушивания – ожидания поступления входящих запросов. Процесс начала сеанса TCP состоит из трёх шагов.

1. Клиент, который намеревается установить соединение, посылает серверу сегмент с номером последовательности и флагом SYN. Сервер получает сегмент, запоминает номер последовательности и пытается создать сокет для обслуживания нового клиента. В случае успеха сервер посылает клиенту сегмент с номером последовательности и флагами SYN и ACK, и переходит в состояние

SYN-RECEIVED. В случае неудачи сервер посылает клиенту сегмент с флагом RST.

2. Если клиент получает сегмент с флагом SYN, то он запоминает номер последовательности и посылает сегмент с флагом ACK. Если клиент одновременно получает и флаг ACK (что обычно и происходит), то он переходит в состояние ESTABLISHED. Если клиент получает сегмент с флагом RST, то он прекращает попытки соединиться. Если клиент не получает ответа в течение 10 секунд, то он повторяет процесс соединения заново.

3. Если сервер в состоянии SYN-RECEIVED получает сегмент с флагом ACK, то он переходит в состояние ESTABLISHED. В противном случае после тайм-аута он закрывает сокет и переходит в состояние CLOSED.

При передаче данных, сегмент, посылаемый от клиентского приложения, содержит в себе АСК и данные, которые планировалось передать. При получении сообщения в ответ всегда отправляется сообщение АСК. Если до получения АСК отправителем истекает тайм-аут, сообщение помещается в очередь на повторную передачу.

Сеанс обмена данными заканчивается процедурой разрыва соединения, которая аналогична процедуре установки, с той разницей, что вместо SYN для разрыва используется служебный бит FIN, свидетельствующий об отсутствии данных для отправки, он устанавливается в заголовке последнего сегмента с данными, отправляемого узлом.

Так как создается кластерная система распределения необходимо обеспечивать надежную доставку сообщений, поэтому был выбран именно протокол передачи данных TCP.

3 Выбор программы для распределенного расчета

В качестве распределённой задачи была выбрана задача поиска пароля по его хэш значению.

Рынок готового программного обеспечения, предлагающий функционал по восстановлению пароля или расшифровки данных достаточно велик. По целевому признаку их можно разделить:

- Программы для сброса, восстановления и анализа паролей ОС Windows или Linux
- Программы расшифровки паролей Wi-Fi по значению handshake.
- Программы для расшифровки персональной информации, хранящейся в браузере.
- Восстановление паролей почтовых клиентов.
- Восстановление паролей к документам.
- Восстановление паролей архивов.
- Другие утилиты

Из вышесказанного программное обеспечение можно поделить на две группы:

- Программы для сброса пароля
- Программы для восстановления пароля

В данной работе нас будет интересовать вторая группа программ. Это обусловлено тем, что сбросить пароль можно в ограниченных случаях и обычно не используются в области распределенных вычислений.

Поэтому рассматривается вторая группа, программы для восстановления пароля. Обычно используется метод перебора пароля, так называемый bruteforce.

Одной из популярных и быстрых программ для восстановления паролей из их хэш-значений является программа Hashcat.

Hashcat — это один из самых быстрых в мире инструментов для восстановления паролей с большим количеством возможностей. Доступны версии как на OS Windows, так и на Linux, есть возможность использования

центрального процессора и использование графического процессора. Hashcat в настоящее время поддерживает огромное количество алгоритмов хеширования.

Hashcat предполагает использование различных типов атак для достижения эффективного покрытия всевозможных хешей:

- Атака брут-форсом (перебором)
- Атака по маске
- Атака перебором всех комбинаций в словаре
- Простой перебор по словарю
- Атака по следу
- Гибридная атака
- Атака перемешиванием
- Атака на основе правила
- Табличная атака
- Атака перебором по заглавным и прописным буквам

Традиционную атаку перебором можно считать устаревшей, в качестве замены разработчики Hashcat рекомендует атаку по маске.

Данная программа была выбрана в качестве распределённой системы по причине того, что она распространяется на бесплатной основе и является обычным консольным приложением. Размер hashcat имеет небольшой размер, что дает удобность использования клиентской части.

Скачать программу можно с сайта https://hashcat.net/hashcat/

Для работы hashcat с OpenCL требуются следующие драйверы:

- Видеокарта AMD в системе Linux требует "RadeonOpenCompute (ROCm)" Software Platform (1.6.180 или более поздняя версия)
- Видеокарта AMD в системе Windows требует "AMD Radeon Software Crimson Edition" (15.12 или более поздняя версия)
- Intel CPUs требует "OpenCL Runtime for Intel Core and Intel Xeon Processors" (16.1.1 или более поздняя версия)
- Intel GPUs в системе Linux требует "OpenCL 2.0 GPU Driver Package for Linux" (2.0 или более поздняя версия)

- Intel GPUs в системе Windows требует "OpenCL Driver for Intel Iris and Intel HD Graphics"
- Видеокарта NVIDIA требует "NVIDIA Driver" (367.x или более поздняя версия)

Hashcat имеет множество опций для запуска, рассмотрим некоторые из них для демонстрации работы программы.

Для проверки работоспособности hashcat, тестирования скорости перебирания паролей, а также правильности устновки нужных драйверов используется опция -b.

Чтобы запустить проверку скорости перебора по самым популярным алгоритмам в командной строке, необходимо выполнить:

.\hashcat64.exe -b

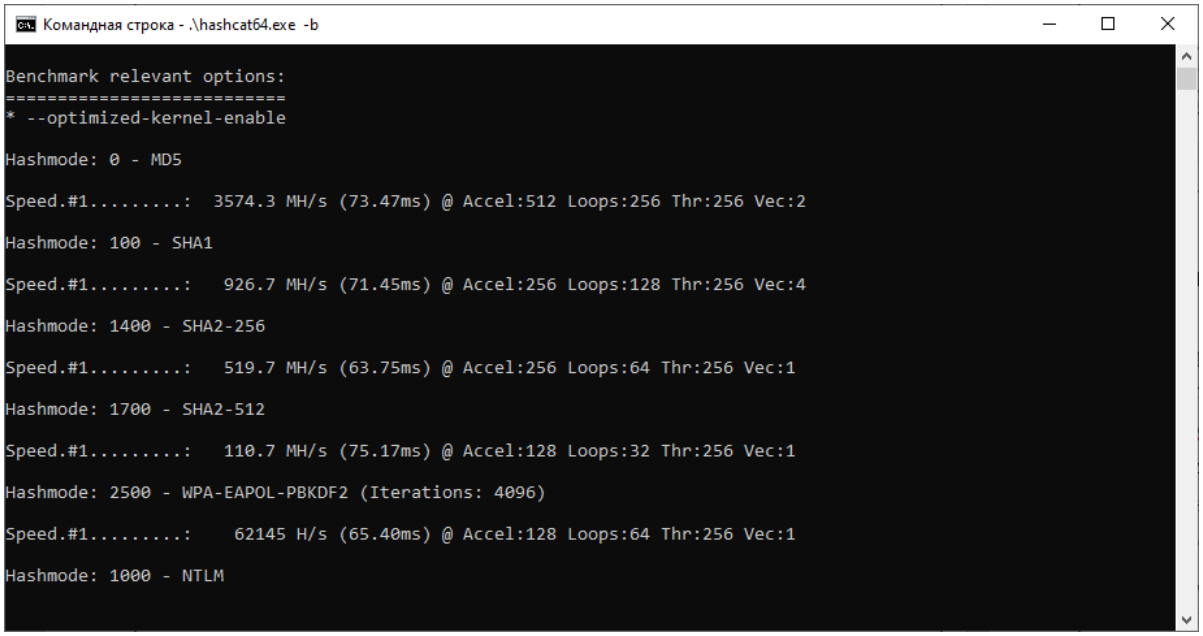


Рисунок 1 - Тестирование hashcat

По результатам теста можно судить о том, что на данной машине все работает, а также оценить скорость перебора на одном вычислительном узле.

Запуск программы выполняется следующим образом:

hashcat [опции]... хеш|файл_хеша|hccapxfile [словарь|маска|директория]…

После имени исполнимого файла через пробел следуют опции. Символ | означает логическое «ИЛИ».

Рассмотрим использование программы на конкретном примере. Допустим нам дан файла handshake с паролем от Wifi-сети зашифрованный с помощью алгоритма WPA2 test.hccapx. Также известно, что зашифрованный пароль состоит только из цифр и имеет длину в восемь символов.

Разные хеши вычисляются по различным алгоритмам. Для того, чтобы правильно запустить атаку в Hashcat, необходимо указать тип атакуемого хеша. Для этого используется опция -m, после которой необходимо указать число, соответствующее выбранному типу хеша. Для WPA2 (WPA-EAPOL-PBKDF2) это -m 2500.

Hashcat, как уже было сказано, поддерживает различные режимы атак, самыми распространёнными являются атака по словарю, которая является устаревшей, и атака по маске.

Для указания режима атаки используется опция -a. С помощью неё можно указывать любой тип атаки, поддерживаемый в Hashcat, самыми популярными является атака по словарю — значение 0, и атака по маске (bruteforce или полный перебор) – значение 3.

Для составления масок используются наборы символов. Наборы символов в Hashcat бывают двух видов:

- встроенные наборы символов
- пользовательские наборы символов

В Hashcat встроенными наборами символов являются:

?l | abcdefghijklmnopqrstuvwxyz

?u | ABCDEFGHIJKLMNOPQRSTUVWXYZ

?d | 0123456789

?h | 0123456789abcdef

?H | 0123456789ABCDEF

?s | !"#$%&'()*+,-./:;<=>?@[\]^_`{|}~

?a | ?l?u?d?s

?b | 0x00 - 0xff

В первой колонке указано условное обозначение набора символов. Для использования в масках, перед данным обозначением необходимо поставить знак вопроса. Например, ?u означает все заглавные буквы, ?d – все цифры.

Допустим, нам известно, что пароль состоит из четырёх символов, первыми двумя символами являются прописные буквы, а третьим и четвёртым символами являются цифры, тогда маска будет следующая:

?l?l?h?h

Чтобы найти пароль из нашего хеша, данного в качестве примера, необходимо составить маску, состоящую из восьми символов, каждый из которых является цифрой. Это маска: ?d?d?d?d?d?d?d?d

.\hashcat64.exe -m 2500 -a 3 test.hccapx ?d?d?d?d?d?d?d?d

Из этого примера видим:

.\hashcat64.exe – исполнимый файл

-m 2500 – опция, устанавливающая тип хеша

-a 3 – опция, значение который запускает атаку по маске

Test.hccapx – файл handshake

?d?d?d?d?d?d?d?d – маска, которая задаёт генерацию паролей длиной в восемь символов, состоящих из цифр.

Бывают ситуации, когда известны некоторые из символов пароля. Например, нам известно, что первыми тремя символами в пароле являются «123». Тогда эти символы можно указать прямо в маске:

123?d?d?d?d?d

Получается команда:

.\hashcat64.exe -m 2500 -a 3 test.hccapx 123?d?d?d?d?d

Известные символы могут находится в любом месте маски – в начале, середине, конце; это могут быть одиночные символы или группы символов.

Из всего описанного можно сделать вывод, что hashcat является гибким инструментом, который можно использовать в распределённой системе, передавая ключи командной строки.

4 Разработка стратегии распределения подзадач

В данной выпускной квалификационной работе бакалавра используется равномерное распределение подзадач — это самый простой для реализации, самый менее трудозатраты вариант, но и наименее оптимальный с точки зрения производительности и использования полосы пропускания сети.

Использовать равномерное распределение подзадач имеет смысл в следующих случаях:

1) Задача имеет чистый поисково-переборный тип и хорошо разбивается на одинаковые подзадачи

2) Подзадачи не требуют большого количества исходных данных и результатов для пересылки по сети

3) Вычислительные узлы, входящие в распределенную систему, являются достаточно идентичными по своей производительности и пропускной способности каналов связи

4) Число вычислительных узлов достаточно велико

Сочетание данных факторов позволяет поделить задачу на некоторое количество подзадач, выбранное после исследования либо с учетом удобства расчета.

Например, если мы решаем задачу восстановления пароля из хэш-значения, и знаем, что пароль состоит из 8 цифровых символов, а наша распределенная система состоит из 10 вычислительных узлов примерно равной производительности, то разумно поделить весь диапазон чисел от 00000000 до 99999999 на 10 равных частей и выдать каждому вычислительному узлу соответствующую подзадачу:

1. 0?d?d?d?d?d?d?d
2. 1?d?d?d?d?d?d?d
3. 2?d?d?d?d?d?d?d и т.д.

Следует отметить, что при наличии отказов узлов производительность системы с таким делением на подзадачи может сильно пострадать.

5 Реализация клиент-серверного приложения

В данной главе описана реализация клиентской и серверной частей приложения. Особое внимание уделяется выделению и описанию различных сущностей, и их взаимодействию между собой. При этом освещены основные особенности реализации, связанные с программированием на выбранном языке. Целью данного раздела является объяснить принцип работы и внутреннее устройство системы в целом.

5.1 Описание работы клиент-серверного приложения

Клиент-серверное приложение включает в себя сервер и клиент. Сервер – это программа с графическими интерфейсом, предназначенный для осуществления взаимодействия с пользователем для исполнения его команд. Клиент – это программа с графическим интерфейсом с запускающийся программой hashcat в фоновом режиме. Клиент и сервер осуществляют общение через сокеты при помощи протокола передачи данных TCP.

5.1.1 Описание работы клиента

Для подключения к серверу, пользователь указывает его IP-адрес и порт, после чего при нажатии на специальную кнопку производится попытка подключения. В случае успеха, клиент переходит в режим ожидания команды от сервера. При неудачном подключении выводится соответствующее информационное сообщение.

При получении команды от сервера, клиент в зависимости от команды в фоновом режиме запускает программу для поиска паролей, либо закрывает ее.

При отключении от сервера, происходит закрытие программы hashcat и отключение от сервера, для дальнейшей работы необходимо заново подключиться к серверу с использованием тех же данных, либо введя новые IP-адрес и порт и нажав на соответствующую кнопку.

5.1.2 Описание работы сервера

Сервер начинает работу так же, как и клиент при нажатии на кнопку.

Сервер работает в асинхронном режиме, при подключении клиента операция подключения клиента блокируется, создается новый сокет для общения с клиентом и начинается асинхронная операция по принятию попытки входящего подключения. Далее производится добавление подключенного клиента в очередь ожидающих задачу, после чего снимается блокировка. Сервер может выполнять следующие действия:

Добавление задачи. Осуществляется попытка добавления задачи в список, а также разделение этой задачи на подзадачи и добавления в отдельный список, который пользователь увидит, только при открытии задачи. В случае возникновения ошибки сервер уведомляет о ней пользователя.

Удаление задачи. Производится удаление задачи из списка задач, удаления всех подзадач этой задачи. Если в момент удаления эта задача была в работе, сервер отправит команды клиентам которые работали над подзадачами, на закрытие программы перебора, добавит клиентов обратно в очередь ожидающих и очистит очередь, если в ней были подзадачи удаляемой задачи.

Открытие задачи. При команде пользователя на открытие задачи, открывается отдельное окно со всей информацией о задаче и список подзадач.

Выгрузка данных. При команде пользователя на выгрузку, приложение формирует Excel таблицу с подзадачами, после чего пользователь может ее сохранить в выбранное им место.

Расшифровка задачи. Сервер добавляет в задачу в очередь, после чего берет первую задачу из очереди и добавляет ее подзадачи в очередь подзадач. После чего из очереди подзадач достается одна подзадача, а из очереди клиентов один клиент, и производится отправка этой подзадачи клиенту.

Выключение сервера. Если подключенных клиентов нет, то закрывается сокет, если были подключенные клиенты, закрываются сокеты каждого из клиентов.

Отключение от сервера. Если клиент по какой-либо причине отключается от сервера, его нерешённая подзадача возвращается в очередь.

На основе описания работы клиента и сервера можно составить UML-диаграмму последовательности (рисунок 2). Она визуализирует действия, описанные выше.

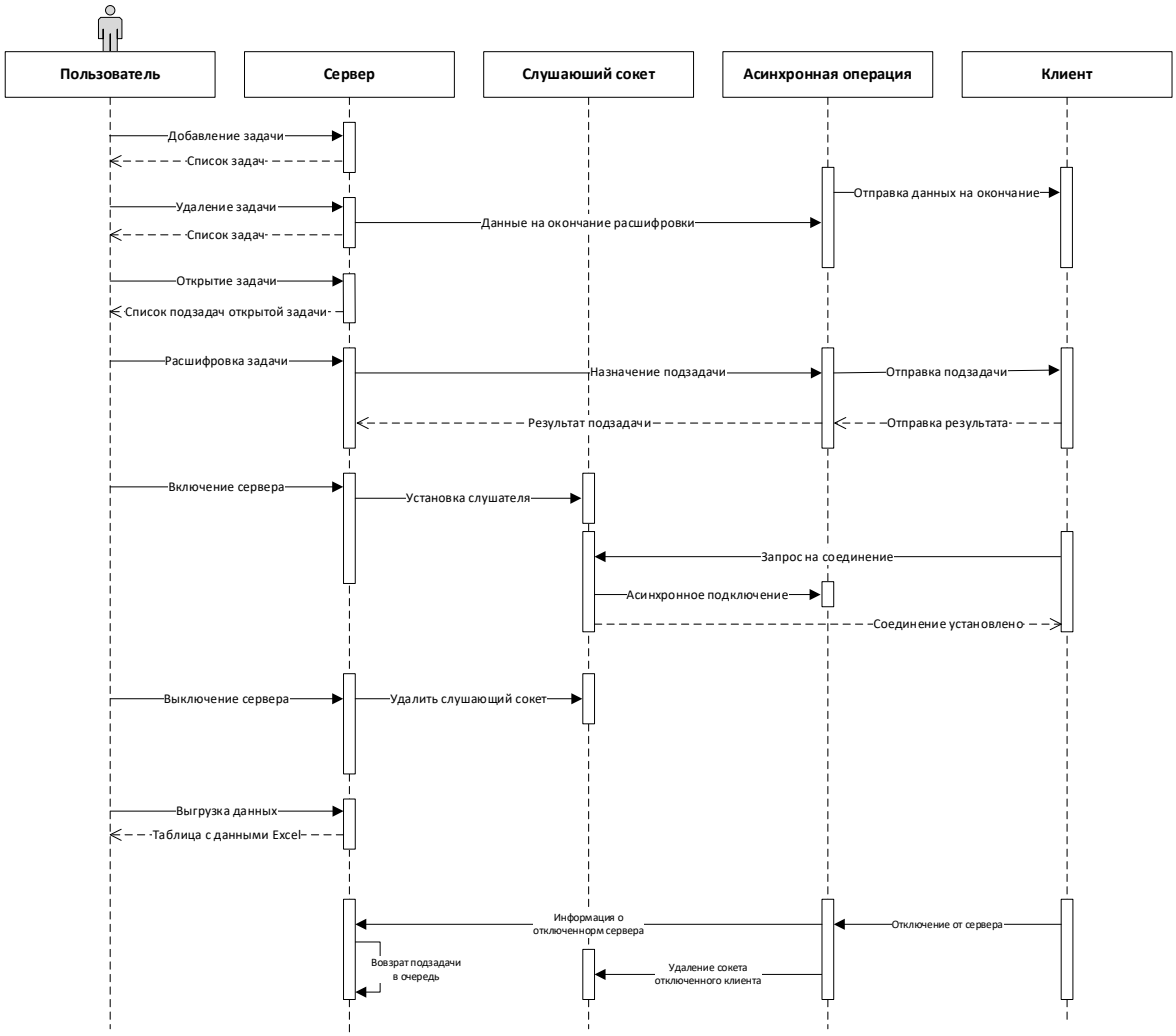


Рисунок 2 - UML-диаграмма последовательности

5.2 Разработка серверной части приложения

В большинстве случаев серверное приложение работает в фоновом режиме без вмешательства пользователя. В данном проекте происходит наоборот – сервер активно взаимодействует с пользователем. Поэтому он имеет удобный графический пользовательский интерфейс, в котором пользователь может добавить задачу, удалить или открыть существующую задачу, включить и

выключить сервер и выгрузить подзадачу в таблицу. На основе вышеизложенного можно построить UML-диаграмму деятельности сервера, представленную на рисунке 3.

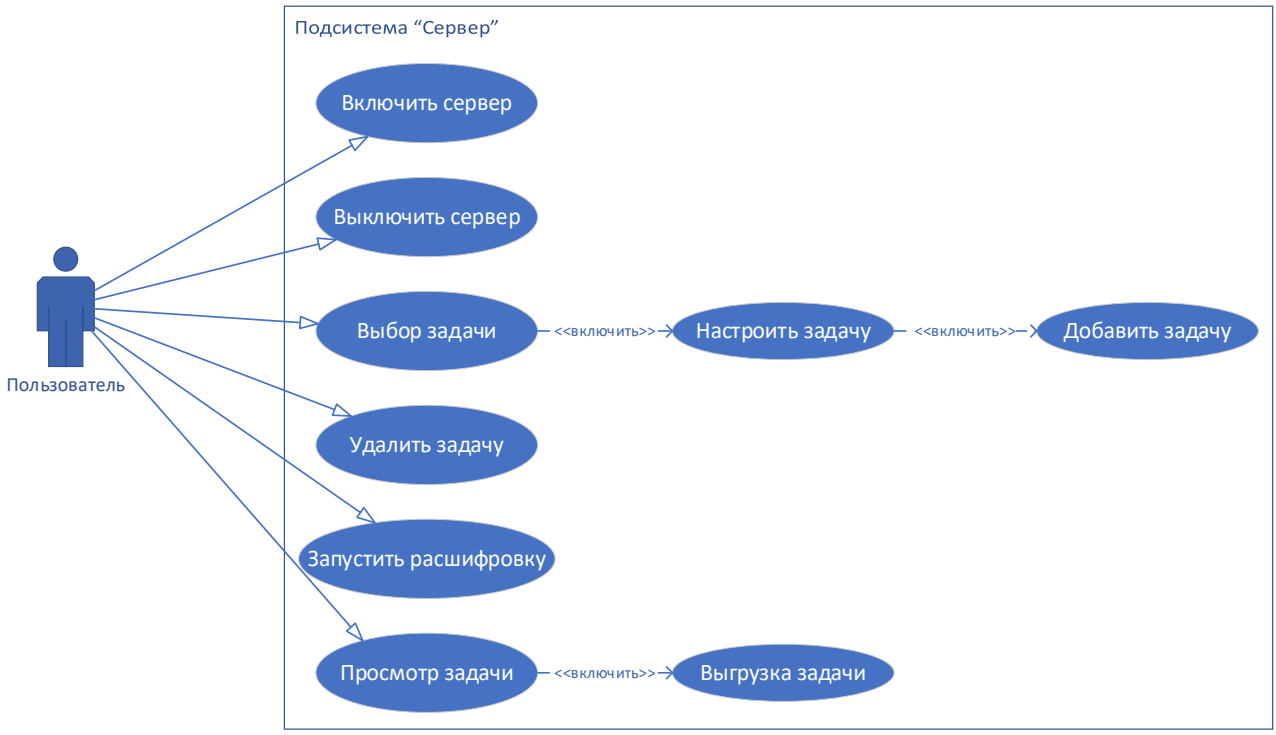


Рисунок 3 - UML-диаграмма вариантов использования сервера

Разрабатываемый сервер состоит из нескольких подсистем. Каждая из них выполняется в отдельной нити. Следует более детально описать работу каждой из них.

Сервер начинает свою работу после команды пользователя на включение сервера. Создается асинхронный слушающий сокет, который ждет подключения клиентов по IP-адресу и порту. Как только подключается новый клиент, он добавляется в очередь ожидающих клиентов, при подключении обработчик запрашивает блокировку, чтобы не допустить ошибок при одновременно подключении нескольких клиентов, по завершению подключения, блокировка снимает и сервер продолжает ожидать подключение следующих клиентов.

Сервер заканчивает работу, после команды пользователя на выключение, очищается все очереди и закрываются сокеты.

Команда на добавление задачи запускает нить данной команды, в которой происходит добавление задачи в список задач, разделение задачи на подзадачи и добавление в список, все операции происходят так же при помощи блокировок, чтобы не допустить ошибок.

Команда удаления запускает нить на удаление задачи, если задача находится в работе, клиентам работающих над этой командой отправляется команда на закрытие приложения для перебора паролей, после чего клиент добавляется обратно в очередь ожидающих клиентов, после чего происходит удаление подзадач из очереди. Так же, как и в случае, если задача находится не в работе, обработчик запрашивает блокировку, происходит удаление самой задачи и подзадач и по завершении снимает блокировку.

Команда открытия задачи отрывает отдельную форму с информацией и подзадачами выбранной задачи, после чего пользователь может просмотреть и закрыть форму, либо выгрузить данные в файл Excel.

Расшифровка задачи создает нить, в которой задача добавляется в очередь и начинает ожидать, пока очередь подзадач не станет пустой. Когда очередь подзадач становится пустой, подзадачи первой в очереди задачи добавляются в очередь подзадач, после чего из очередей клиентов и подзадач достается по одному элементу. Подзадача и файл handshake сериализуются, выполняется отправка сериализованных данных клиенту и начинается ожидание ответа от клиента так же в виде сериализованных данных с клиента, которые десериализуются, и выводится ответ. Нить работает до тех пор, пока не закончится очередь задач.

При отключении клиента, который выполнял расшифровку, по его выданному при подключении GUID-идентификатору, сервер находит подзадачу и возвращает ее в очередь.

На основе вышеизложенного можно построить UML-диаграмму деятельности сервера, представленную на рисунке 4.

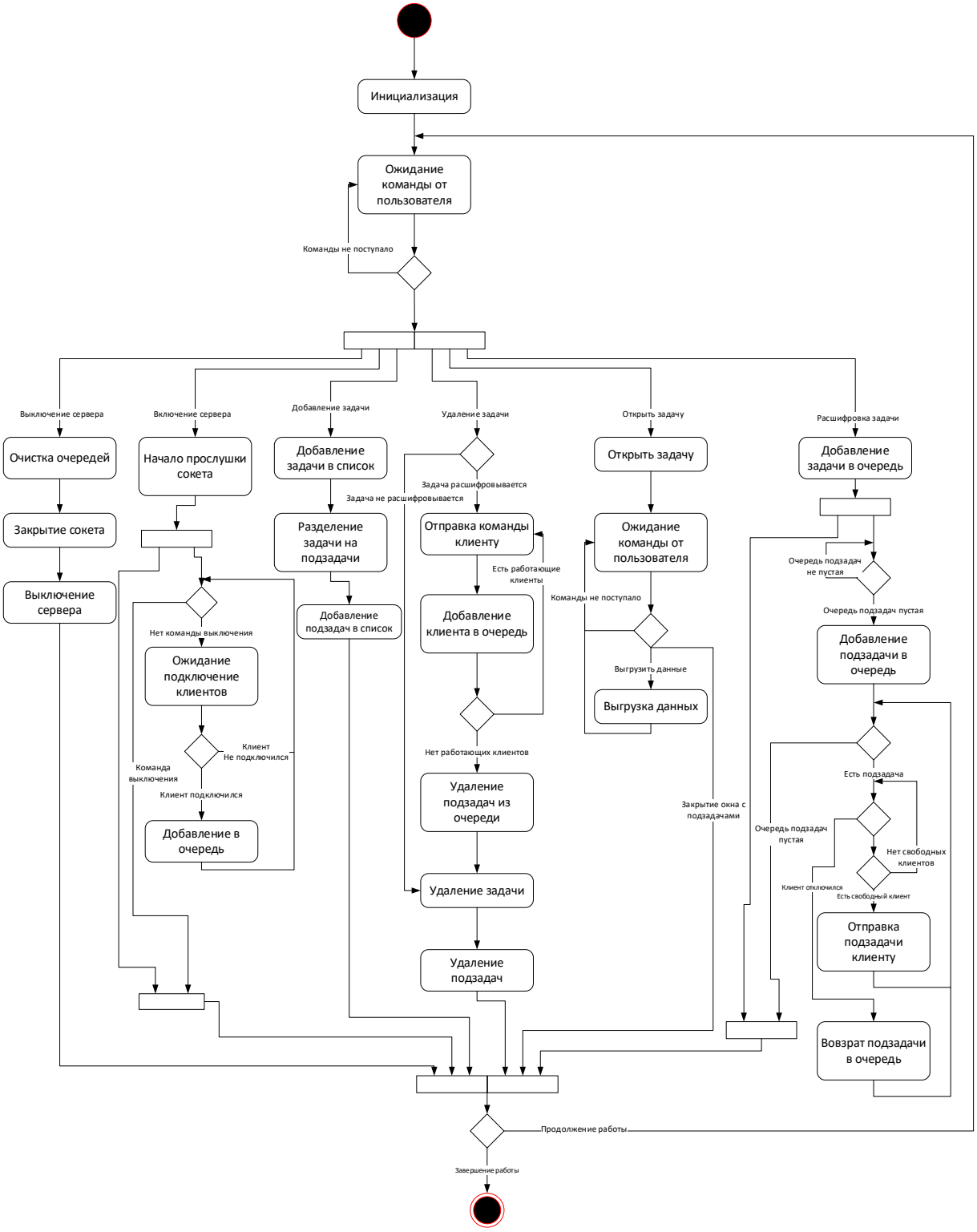


Рисунок 4 - UML-диаграмма деятельности сервера

Из диаграммы деятельности и описания работы можно выделить следующие состояния сервера:

1. Состояние ожидание команды от пользователя. В этом состоянии сервер ждет команды от пользователя.

2. Состояние ожидания клиента. В этом состоянии сервер ожидает подключения нового клиента.

3. Состояние добавления задачи. В этом состоянии сервер обрабатывает запрос на добавление новой задачи.

4. Состояние удаление задачи. В этом состоянии сервер обрабатывает запрос на удаление существующей задачи.

5. Состояние открытие задачи. В этом состоянии сервер осуществляет открытие формы с информацией о задаче.

6. Состояние расшифровки задачи. В этом состоянии сервер осуществляет отправку задания клиенту, ожидание ответа и при необходимости возврат подзадачи в очередь.

Сервер разрабатывается на языке программирования C#. C# является объектно-ориентированным языком программирования, для описания сущностей в нем используются классы. Их использование позволяет создавать комплексные системы со сложной логикой. Рассмотрим классы, которые понадобятся для реализации клиента.

Form1 – это основной класс сервера. Данный класс отвечает за внешний вид главного экрана клиента. На нем расположены основные элементы взаимодействия пользователя с клиентом.

addtask – это класс со списками задач и подзадач, который добавляет, удаляет и разбивает на подзадачи добавленную задачу.

Client – это класс подключенного клиента.

DataTask – это класс отдельного открывающегося окна с информацией задачи и списком подзадач.

GetIP – это класс получение локального адреса машины.

MyQueue – это класс очереди ожидающих задание клиентов.

Outfile – это класс формирования таблицы Excel и выгрузка ее на машину.

Queuetask – это класс очереди работающих задач.

Queuezadach – это класс очереди подзадач работающей задачи.

Qz – это класс с полями для очереди подзадач.

Raspredtask – это класс со свойствами и полями для подзадач.

Spisoktask – это класс со свойствами и полями для задач.

DatainClient – это класс из библиотеки классов со свойствами и полями для сериализации отправляемых данных клиенту.

DatainServer – это класс из библиотеки классов со свойствами полями для десериализации принятых данных от клиента.

На рисунке 5 изображена UML-диаграмма классов сервера. Она показывает связи между классами.

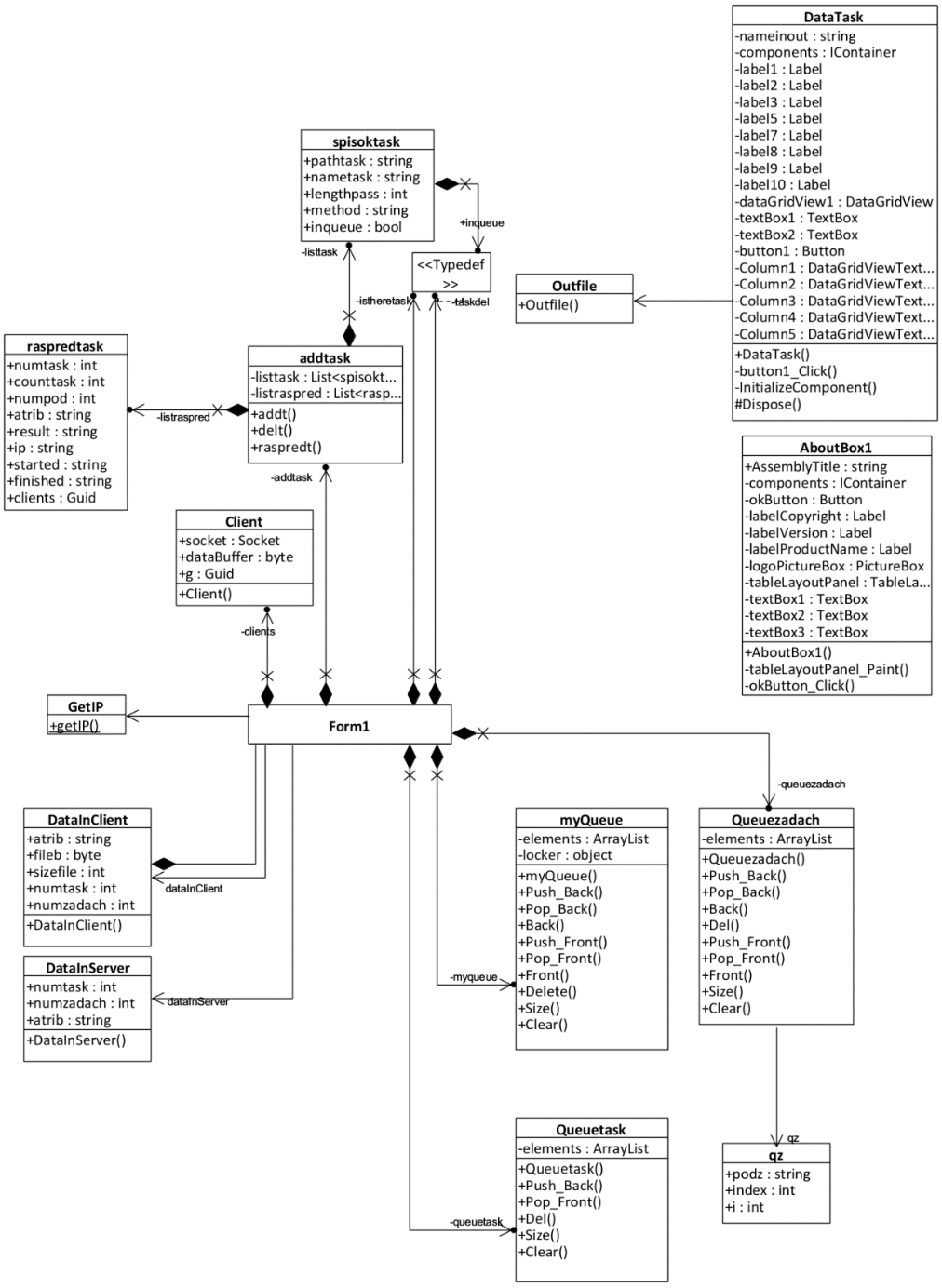


Рисунок 5 - UML-диаграмма классов сервера

5.3 Разработка клиентской части приложения

В большинстве случаях участие пользователя необходимо при работе с клиентским приложением. В данном проекте наоборот - клиент является той частью приложения, в которой взаимодействие с пользователем минимальна. Поэтому он должен иметь минимальный и удобный графический интерфейс для подключения к серверу. Остальной функционал клиента работает в фоновом

режиме, не мешая клиенту. На основе вышеизложенного можно построить UML-диаграмму деятельности клиента, представленную на рисунке 6.

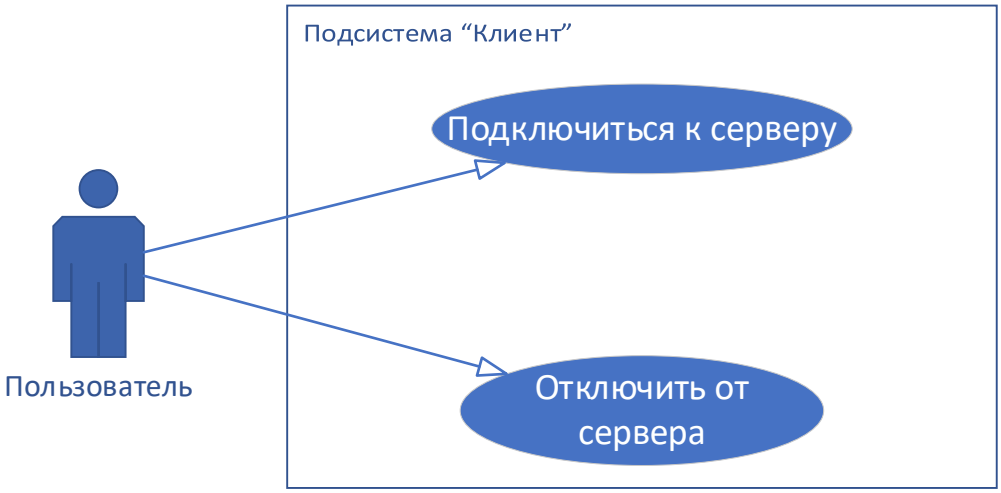


Рисунок 6 - UML-диаграмма вариантов использования клиента

Разрабатываемый клиент состоит из двух основных подсистем. Первая отвечает за подключение и обработку запросов сервера. Вторая открытие и закрытие программы расшифровки hashcat. Каждая из них выполняется в отдельной нити.

Как только пользователь ввел данные для подключения и клиентское приложения успешно подключилось к серверу, начинается процесс ожидания данных.

При получении данных от сервера, который приходят в сериализованном виде, клиентское приложение десериализует их и запускает в отдельной нити в фоновом режиме программу для перебора пароля hashcat, нить работает до тех пор, пока от сервера не придет команда на закрытие перебора, клиент не отключится от сервера или перебор закончится. Если закрытие нити с hashcat не произошло, и он закончил работу, ответ от приложения перебора паролей и данные о подзадачи сериализуются и отправляются на сервер. На основе вышеизложенного можно построить UML-диаграмму деятельности клиента, представленную на рисунке 7.

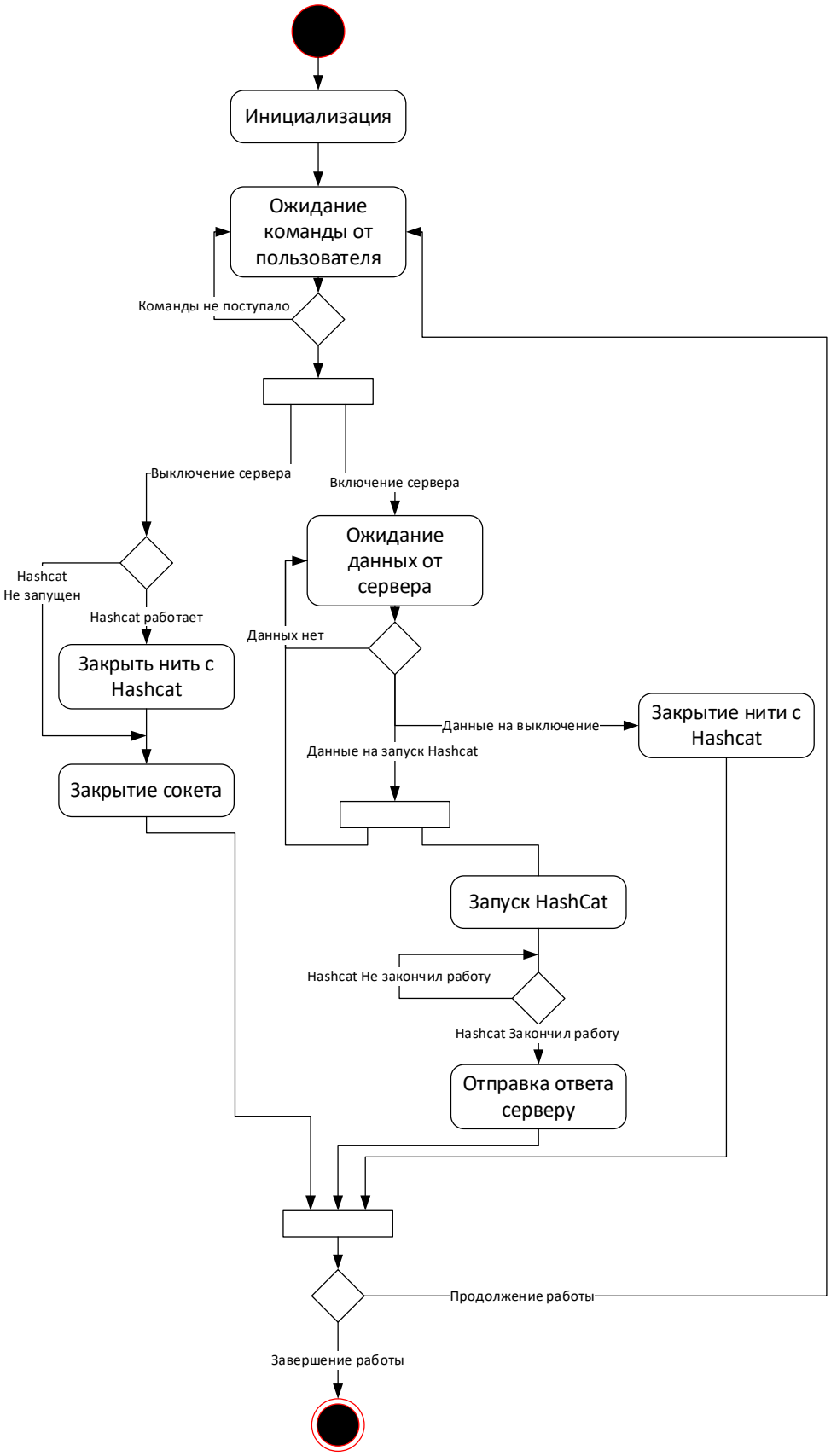


Рисунок 7 - UML-диаграмма деятельности клиента

Из диаграммы деятельности и описания работы можно выделить следующие состояния клиента:

1. Состояние подключение к серверу. В этом состоянии клиент пытается подключиться к серверу и получить список задач.

2. Ожидание команды от пользователя. В этом состоянии клиент ожидает получения команды от пользователя.

3. Состояние обработка запроса. В этом состоянии сервер выполняет действия, запрашиваемые сервером.

Клиент так же, как и сервер разрабатывается на языке программирования C#. Рассмотрим классы, которые понадобятся для реализации клиента.

Form1 – это основной класс клиента. Данный класс отвечает за внешний вид главного экрана клиента. На нем расположены основные элементы взаимодействия пользователя с клиентом.

SocketPacket – это класс со свойствами и полями для подключения.

WorkHash – это класс, который взаимодействует с программой перебора пароля hashcat.

DatainClient – это класс из библиотеки классов со свойствами и полями для десериализации принятых данных от сервера.

DatainServer – это класс из библиотеки классов со свойствами полями для сериализации отправляемых данных серверу.

На рисунке 8 изображена UML-диаграмма классов клиента. Она показывает связи между классами.

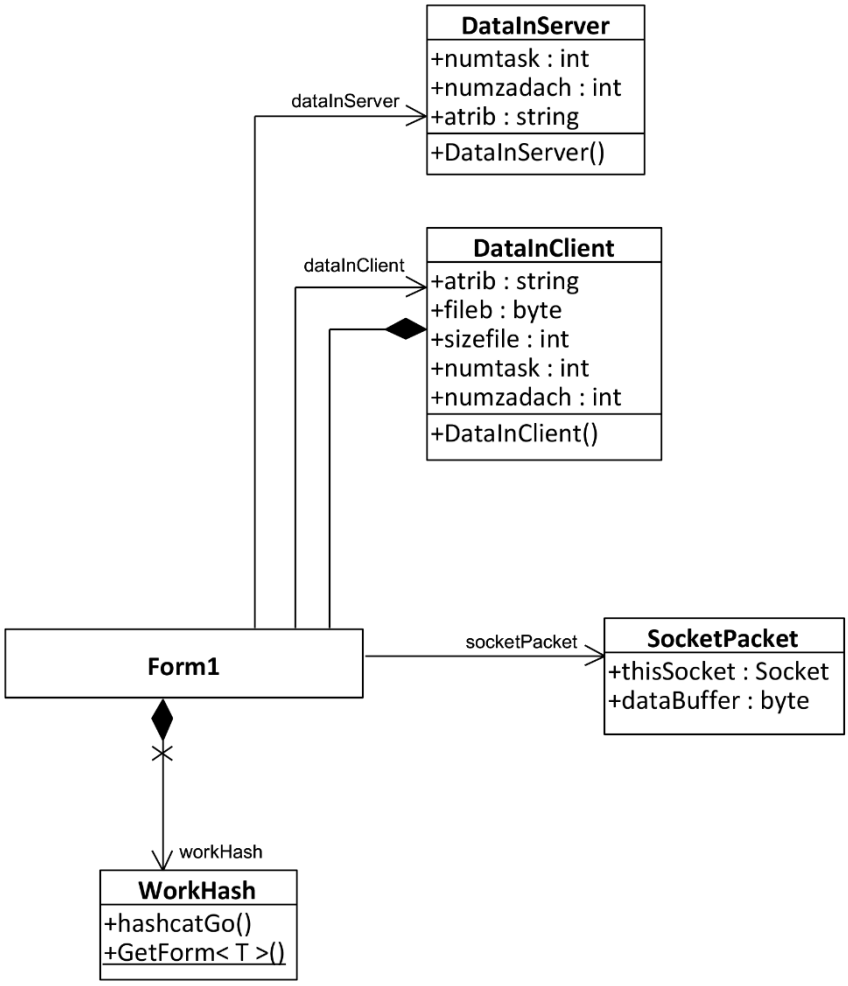


Рисунок 8 - UML-диаграмма классов клиента

6 Описание приложения

6.1 Руководство пользователя сервера

После запуска сервера перед пользователем предстает главное окно программы (рисунок 9). На нем расположены кнопки включения, выключения сервера, выбор файла handshake, добавления задачи, расшифровки задачи и удалении задачи. Так же перед добавлением задачи, в специальном окне можно настроить задачу, выбрать длину пароля (от 8 до 10), выбрать символы перебора для маски: только буквы (строчные), только цифры, цифры и буквы(строчные). Большинство нужной информации о действиях пользователя и клиента выводится в окно «Консоль», информация для подключения к серверу показана в полях «Local IP» и «Порт», количество подключенных клиентов показаны в пункте «Кол-во подключившихся».

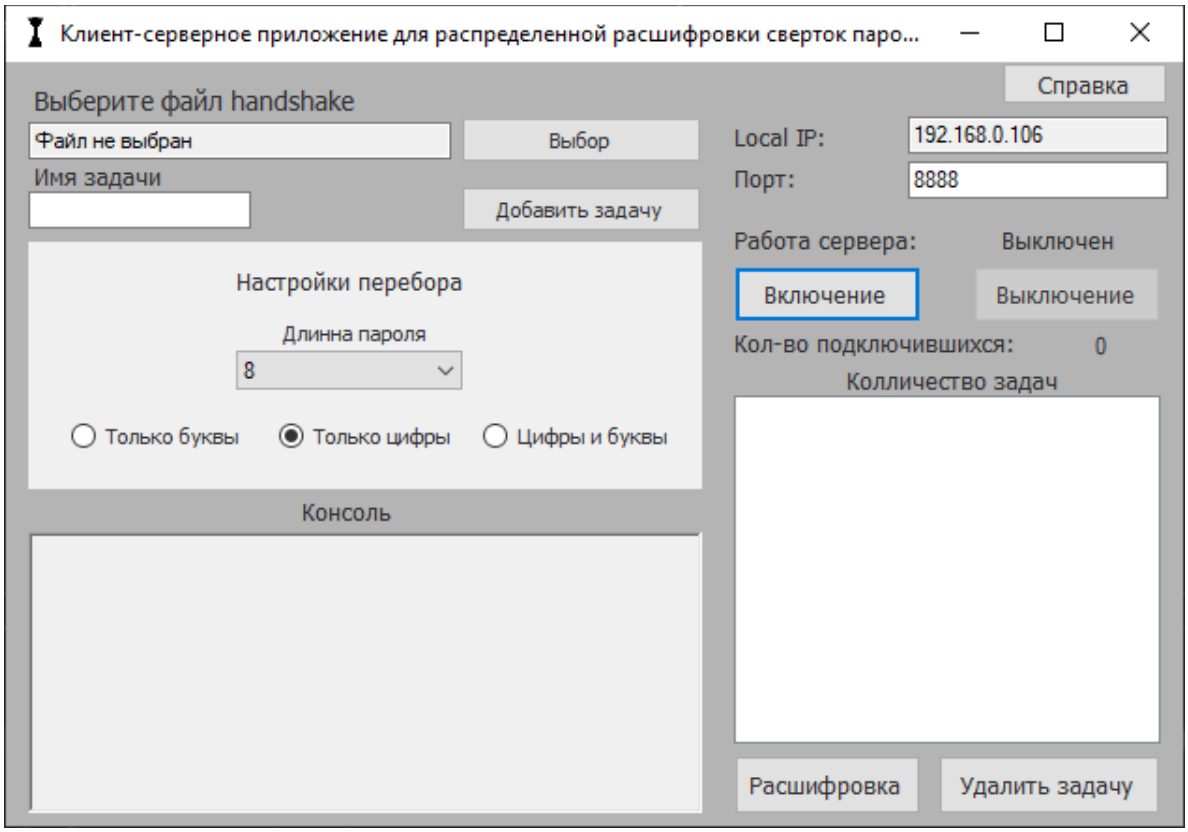


Рисунок 9 - Главное окно сервера

Для создания задачи, в начале необходимо нажать на кнопку «Выбор» и выбрать файл с хэш-значением с расширением hccapx. (рисунок 10).

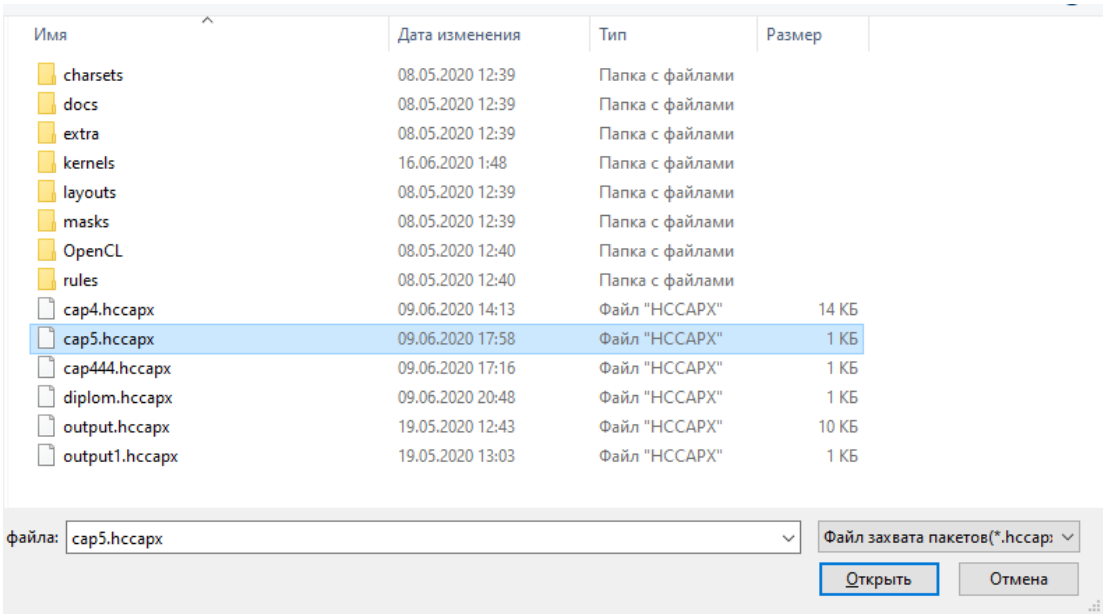


Рисунок 10 - Окно выбора файла handshake

После выбора файла нужно ввести название задачи и нажать на кнопку «Добавить задачу», если все правильно введено, задача добавится в список задач. (рисунок 11)

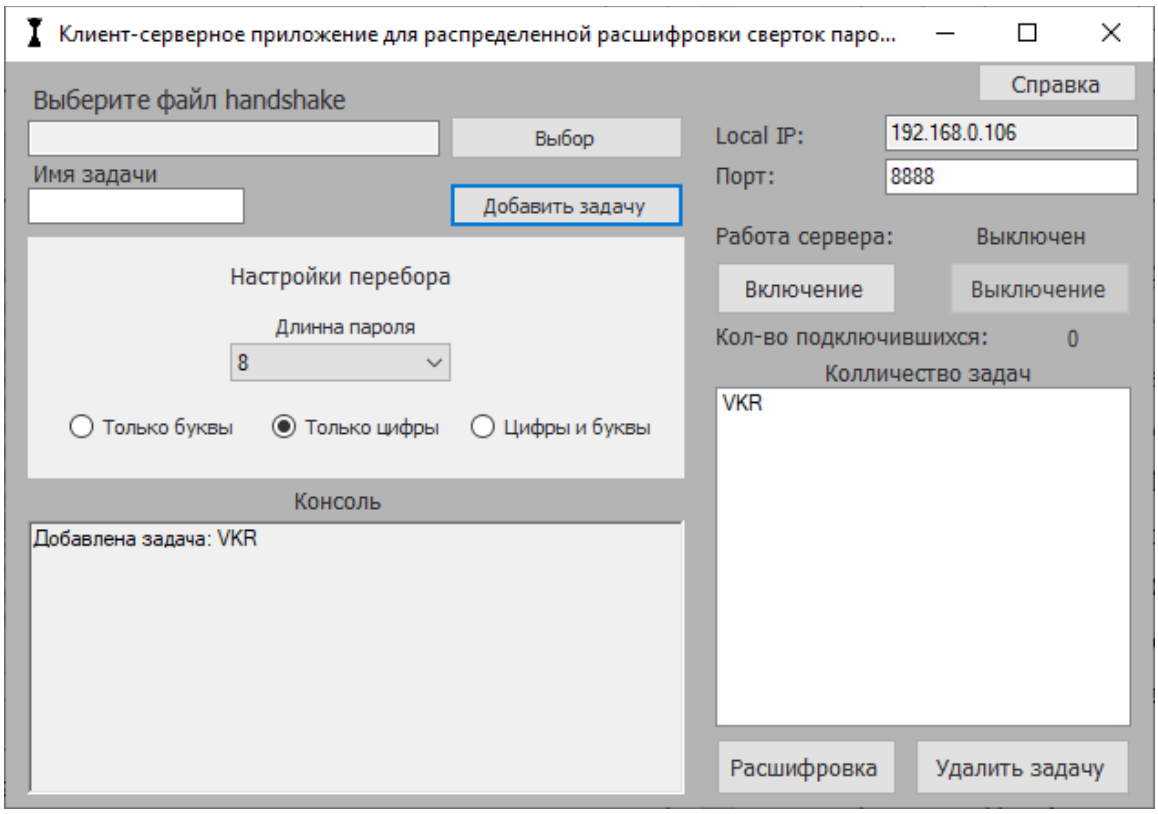


Рисунок 11 - Добавление задачи в список

Если поле “Имя задачи” пустое, либо состоит из более чем 16 символов, появляется сообщение об ошибке (рисунок 12). При вводе в это же поле названия, которое уже есть в списке, появляется сообщение об ошибке (рисунок 13).

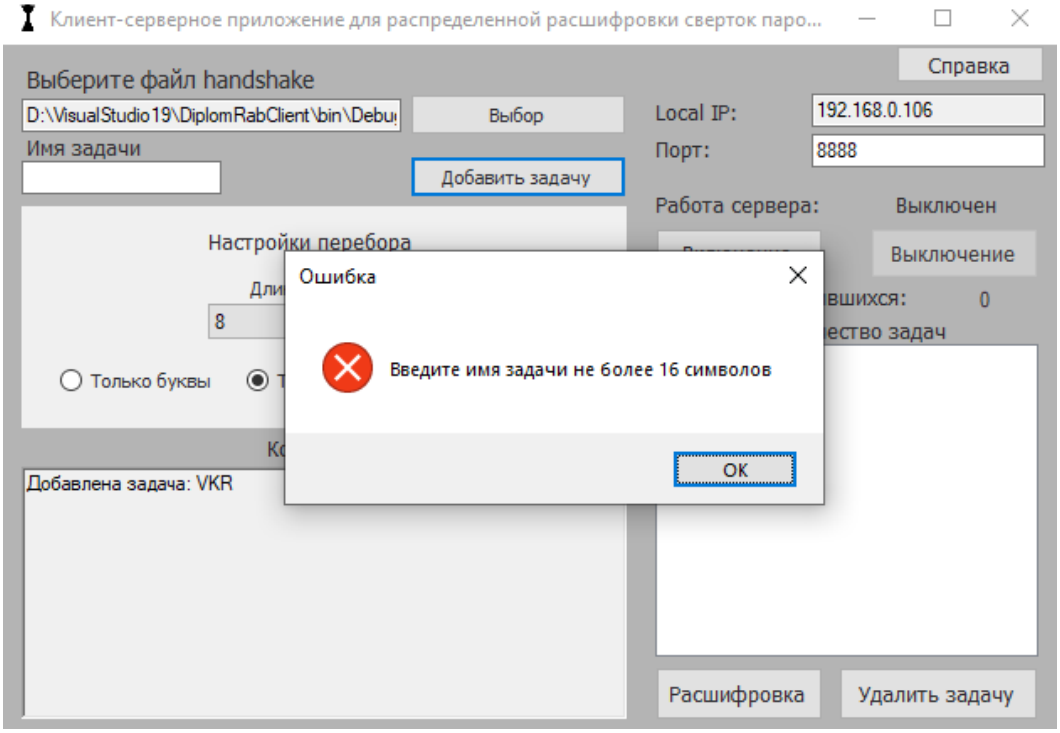


Рисунок 12 - Ошибка при вводе имени задачи

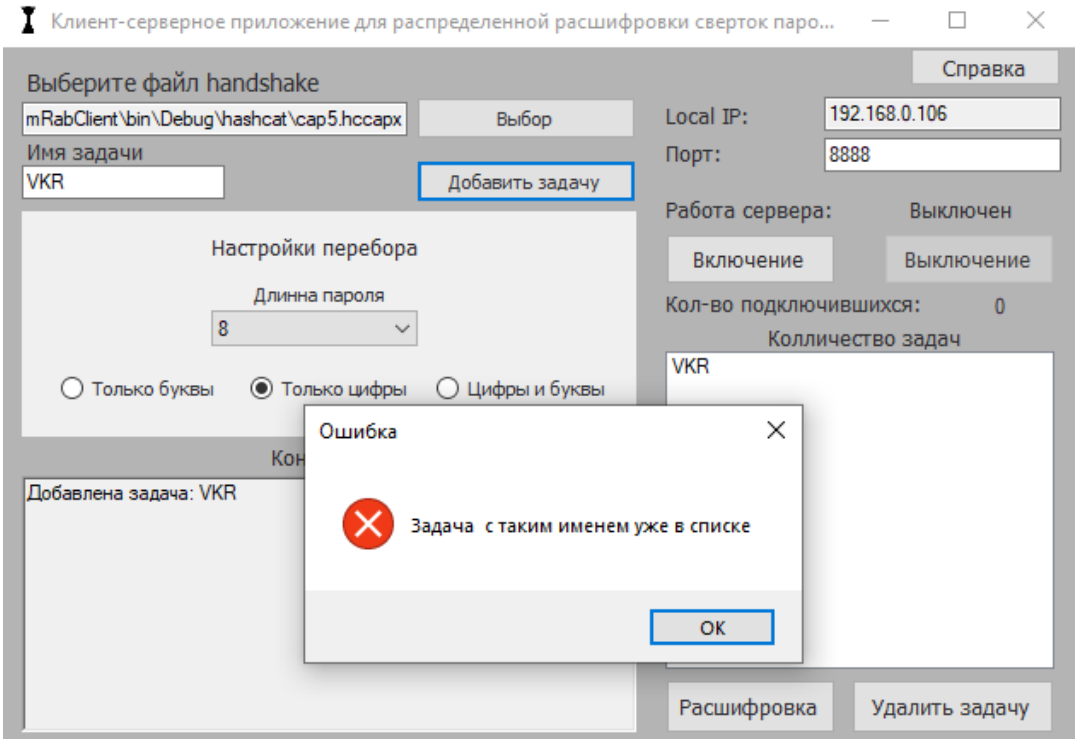


Рисунок 13 - Ошибка при вводе имени задач, которая есть в списке

Если добавлять задачу без выбранного файла, то так же появится сообщение об ошибке (рисунок 14).

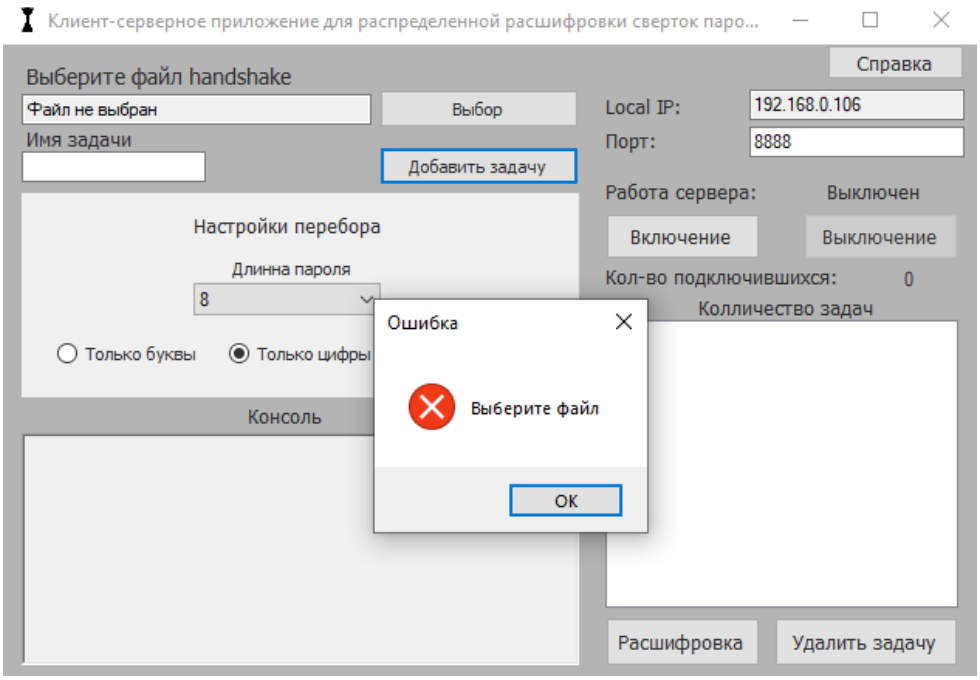


Рисунок 14 - Ошибка, если файл не выбран

Расшифровать и удалить задачу можно при помощи кнопок на интерфейсе, чтобы открыть задачу, нужно сделать двойной клик по задаче. Такие же действия можно сделать при помощи нажатии правой кнопкой мыши на задаче (рисунок 15).

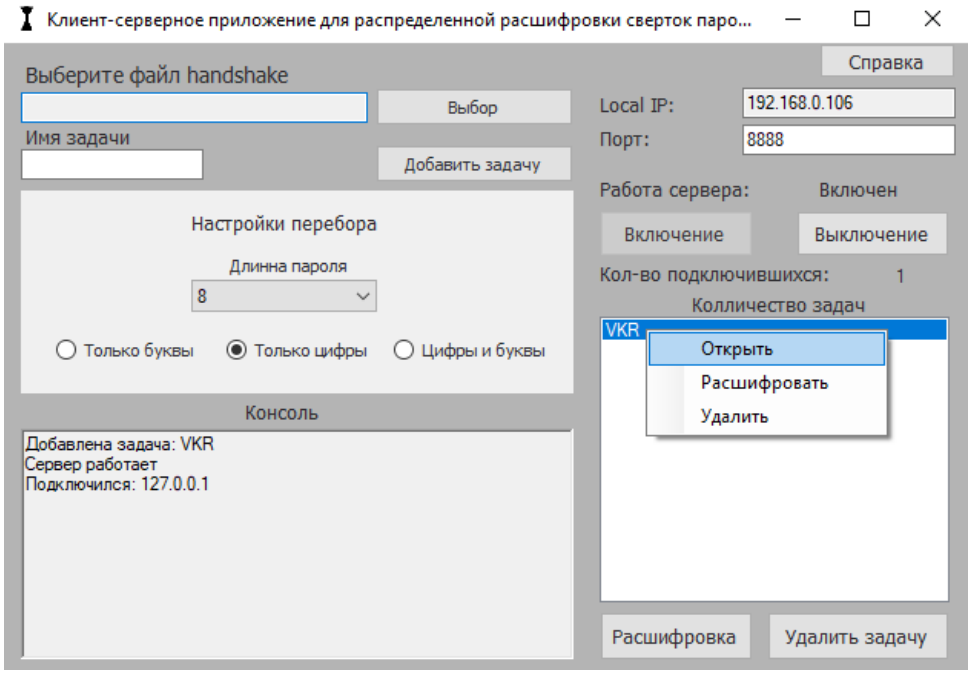


Рисунок 15 – Демонстрация контекстного меню

После добавление задачи, включении сервера и подключении клиента, сообщения об этом выводятся в консоль. При нажатии на кнопку расшифровать, справа от задачи появляется спец. Знак «🕓», демонстрирующий работу задачи, сервер начинает раздавать подключенным клиентам по одной подзадаче.

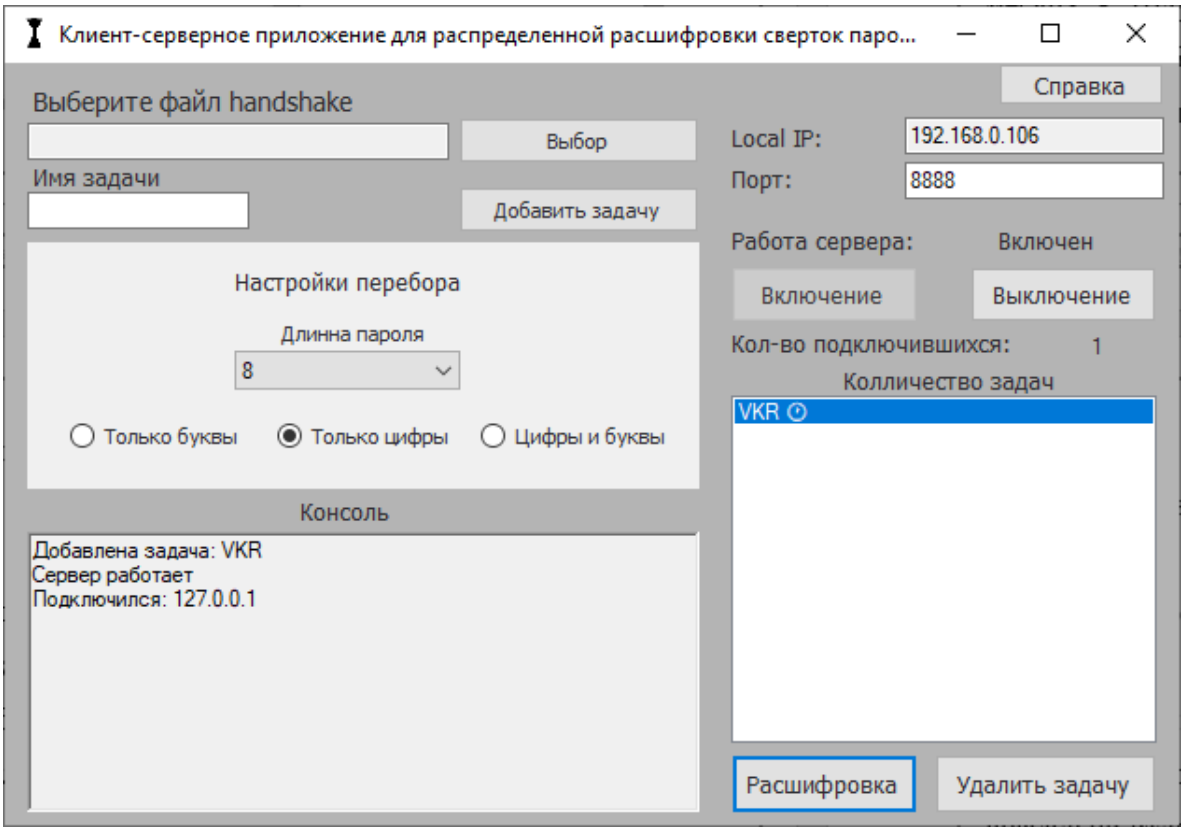


Рисунок 16 - Демонстрация работы расшифровки задачи

Если клиент прислал ответ в виде расшифрованного пароля, справа от задачи спец. Знак «🕓» заменяется на спец. Знак «✔», что означала удачную расшифровку пароля. Задача заканчивает работу (рисунок 17).

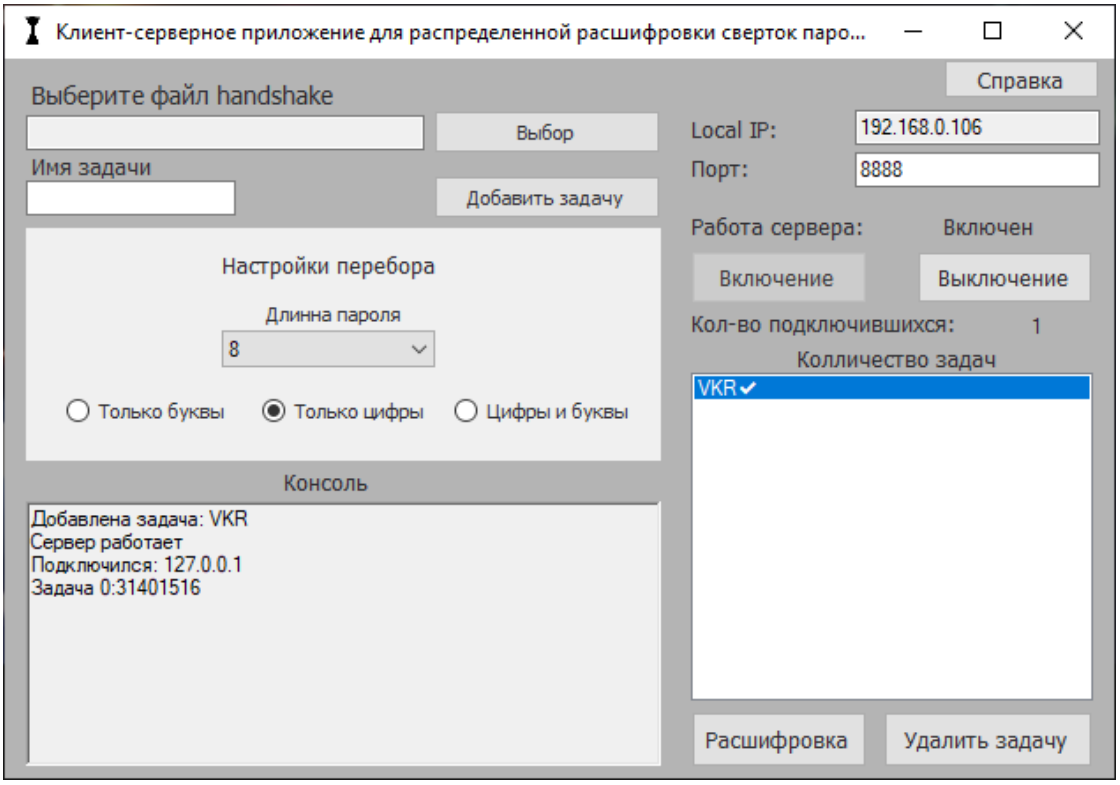


Рисунок 17 - Демонстрация завершенной задаче и удачного поиска пароля

Ответ от клиента выводить так же в «Консоль». Если все подзадачи отработаны и пароль не был найден, спец. Знак «🕐» заменяется на спец. Знак «✖».

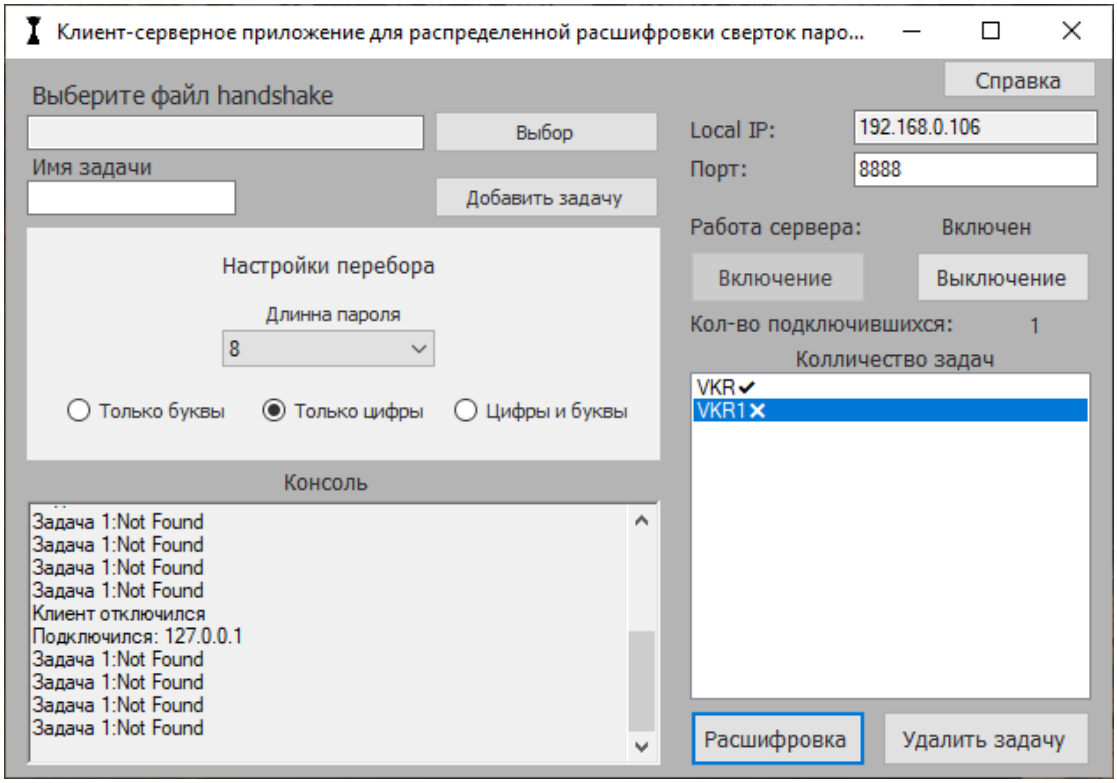


Рисунок 18 - Демонстрация найденного и не найденного пароля

В открытой информации выводится информация о названии задачи, имени файла с хэш-значением, путь файла, длинна пароля, набор символов. В таблице выводится опции запуска hashcat у клиента, результат ответа от клиента, IP-адрес клиента, работающего над этой подзадачи, время отправки подзадачи, время ответа клиента. Кнопка «Выгрузить в таблицу» открывает окно выбора пути для сохранения таблицы в файл Excel. (рисунок 19).

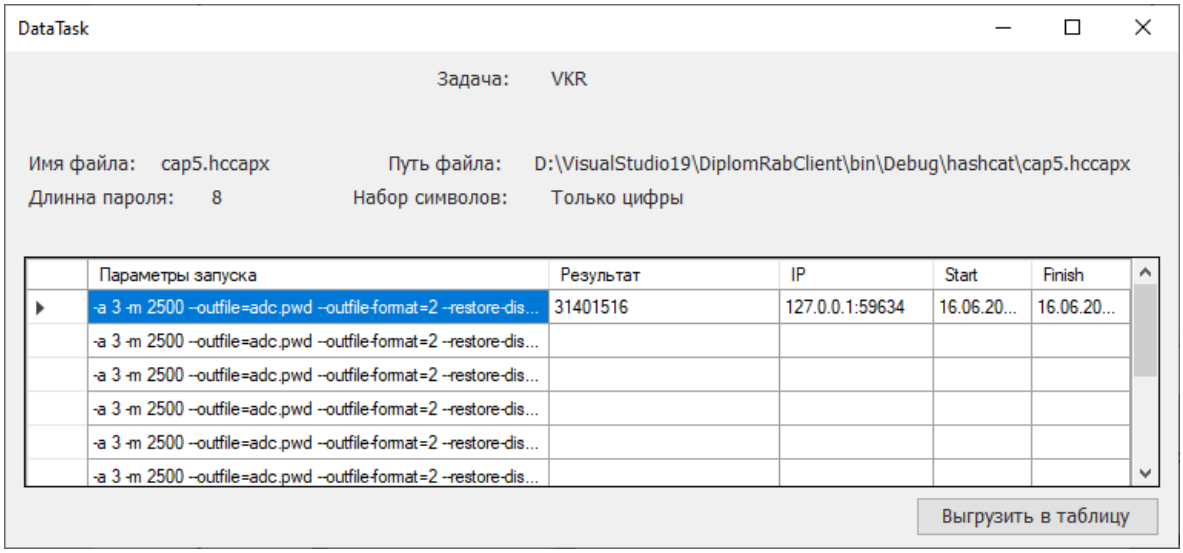


Рисунок 19 - Открытая задача с информацией и подзадачами

Кнопка «Удалить задачу», удаляет все подзадачи и саму задачу из списка задач, после чего выводится сообщение об успешном удалении задачи в «Консоль» (рисунок 20). При нажатии на кнопку «Выключение», сервер закроет все соединения с клиентами и очистит очередь, для продолжения расшифровки задачи, необходимо заново нажать на кнопку «Расшифровка».

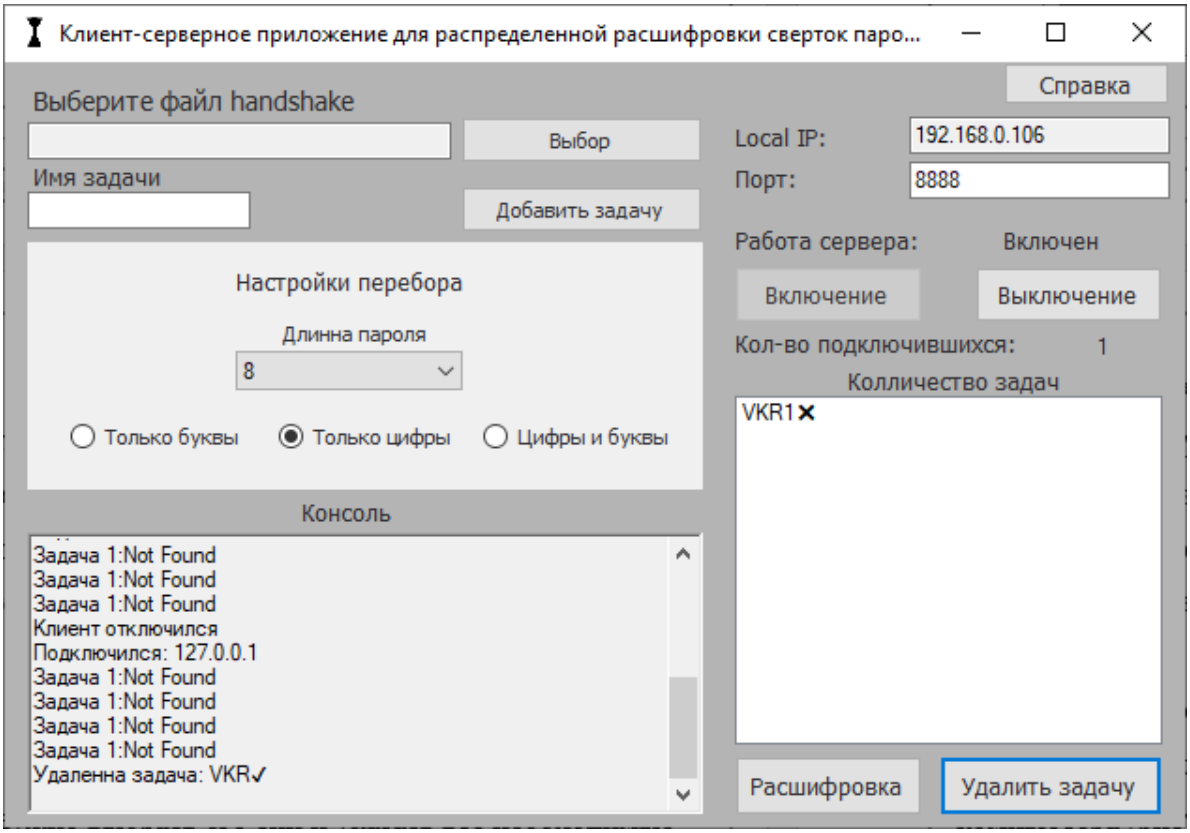


Рисунок 20 - демонстрация удаление задачи

Если клиент работал над решением задачи и в этот момент отключился от сервера не закончив работу над handshake, эта подзадача возвращается обратно в очередь. Первому освободившемуся клиенту будет выдана эта подзадача. (рисунки 21,22)

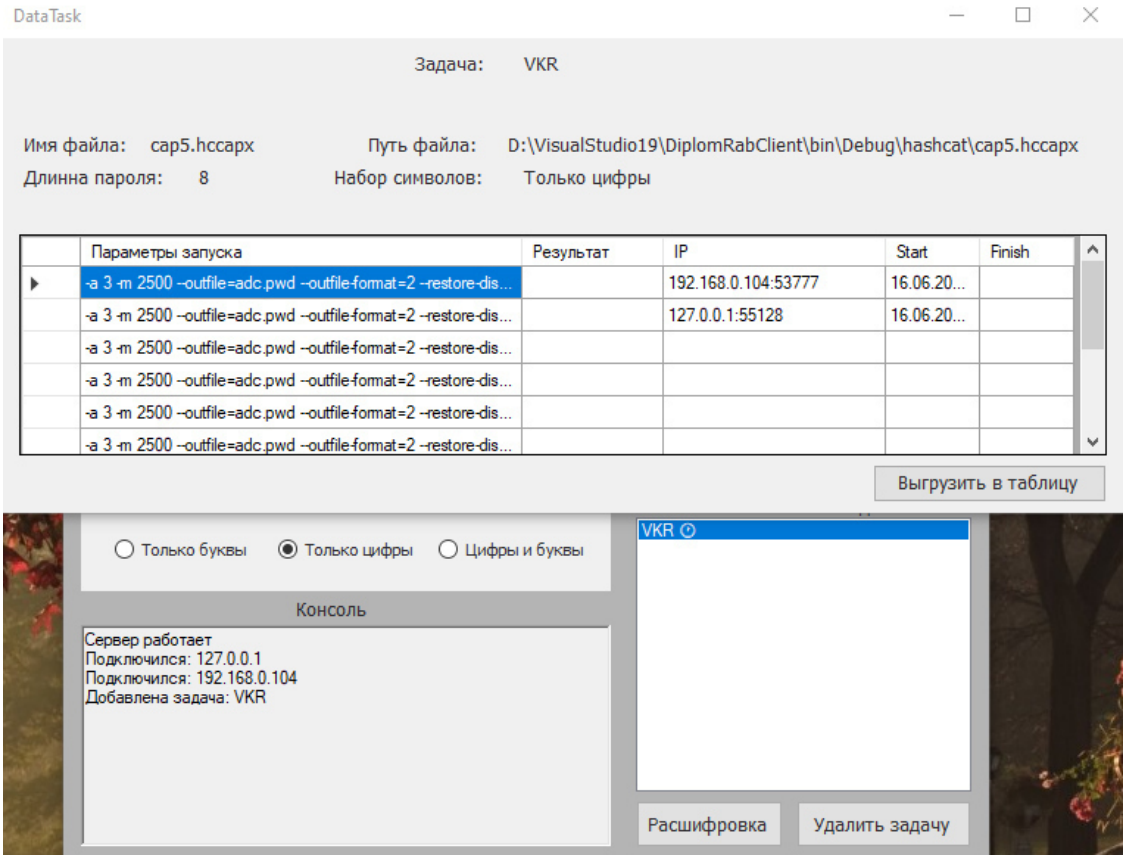


Рисунок 21 - клиенты работающие над подзадачами

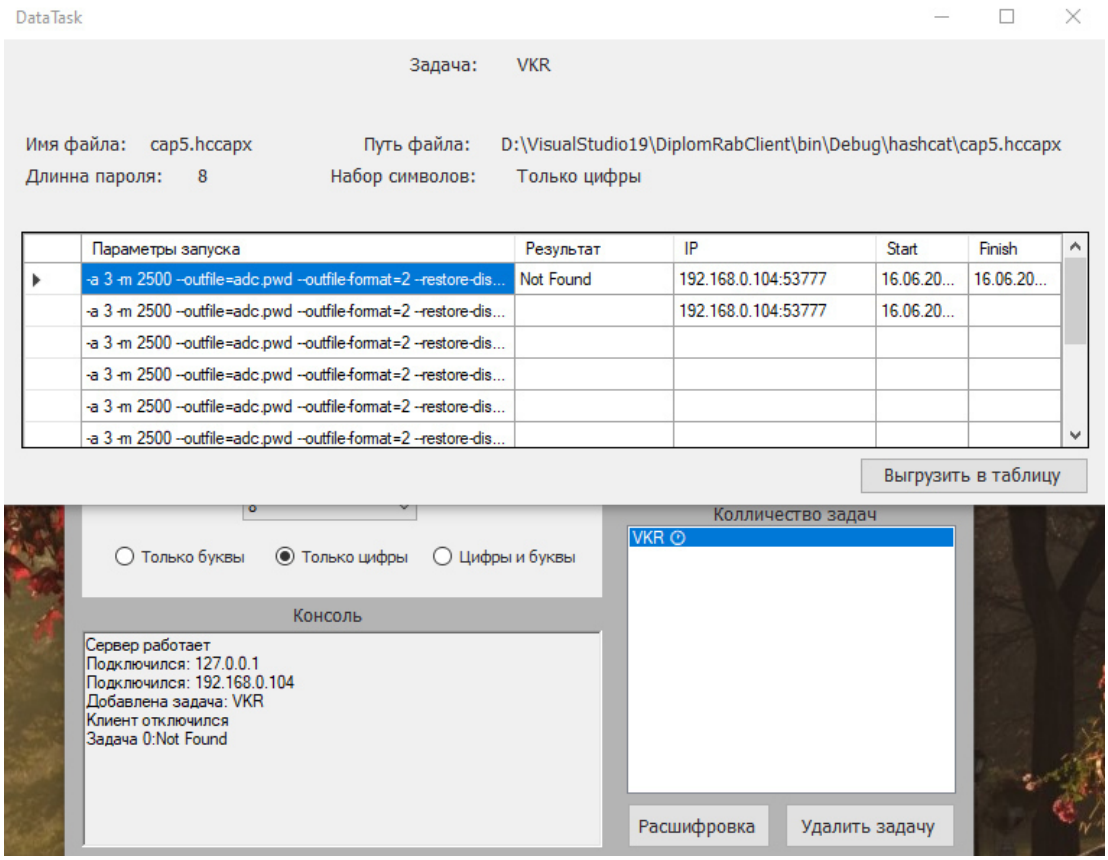


Рисунок 22 - освободившемуся клиенту выдается подзадача отключенного клиента

6.2 Руководство пользователя клиента

После запуска клиента перед пользователем предстает главное окно программы (рисунок 23). На нем расположены 2 поля для ввода «IP-адреса» и «порта» сервера, 2 кнопки «Подключиться» и «Отключиться» для подключения и отключения от сервера соответственно.

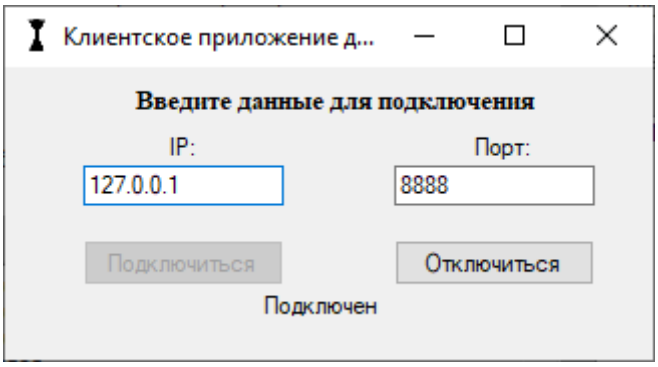


Рисунок 23 - Вид начально экрана клиента

При вводе не правильных данных для подключения и нажатии кнопки «Подключиться», выводиться сообщение об ошибке (рисунок 24).

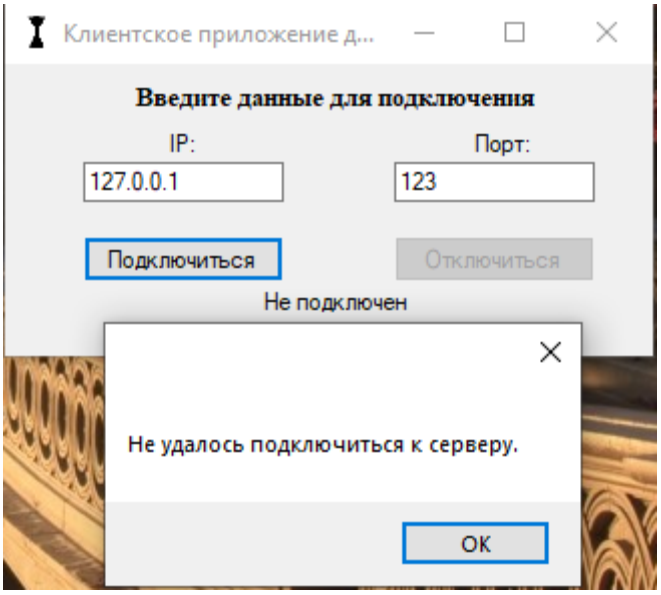


Рисунок 24 - Ошибка подключения к серверу

Клиент может работать в свернутом режиме, не мешая пользователю. При сворачивании клиентского приложения, оно сворачивается в трей, из которого пользователь может развернуть и закрыть программу (рисунок 23).

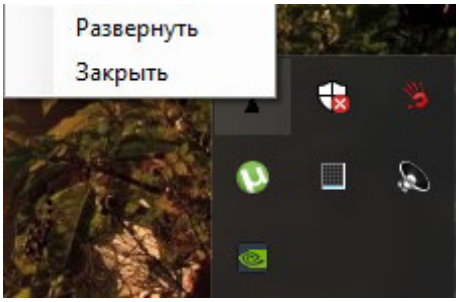


Рисунок 25 - Свернутое в «трей» клиентское приложение

При отключении сервера, клиентское приложение выведет ошибку и будет ожидать действий от пользователя (рисунок 26).

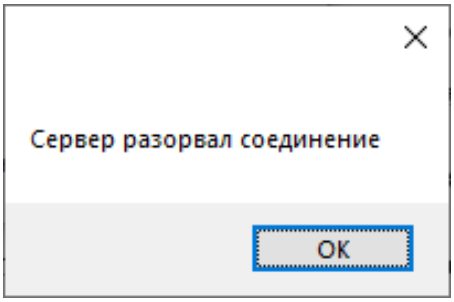


Рисунок 26 - Сообщение о выключении сервера

6.3 Спецификация сервера

Сервер разработан с использованием графического интерфейса Windows Forms. Язык программирования C#, на котором реализован сервер, позволил полностью реализовать весь задуманный функционал приложения.

Класс Form1 — это основной класс сервера

Основные методы и поля класса Form1:

Server – поле являющиеся сокетом.

Myqueue – поле являющиеся очередью ожидающих клиентов.

Queuetask – поле являющиеся очередью запущенных задач.

Queuezadach – поле являющиеся очередь подзадач.

S – поле создания потока, резервным хранилищем которого является память для сериализации данных.

Clients – после хранения подключенных клиентов

Mre – событие останавливающие нить распределения подзадач по клиентам для удаления задачи.

Deltask – метод для создания нити удаление задачи.

Sraspred – метод для распределения подзадач по клиентам.

Gotask – метод инициализации запуска задачи в работу.

Deltask – метод инициализации удаления задачи.

Addtask – поле работы со списком задач и подзадач и разбиение на подзадачи.

Opentask – метод открытия формы с информацией о задаче.

UpdateControls – метод обновления формы при включении и выключении сервера

OnClientConnect – метод обратного вызова, вызываемая при подключении клиента.

WaitForData – метод ожидания данных от подключённого клиента.

OnDataReceived – метод обратного вызова, вызываемая при обнаружении сокетом данных в потоке.

Send – метод отправления данных клиенту.

CloseSockets – метод закрытия сокетов.

Класс addtask – это класс взаимодействия со списками задач и подзадач.

Основные методы и поля класса addtask:

Listtask - это создание списка для объекта spisoktask.

Listraspred – это создание списка для объекта raspredtask.

Addt – метод добавление задачи в список.

Delt – метод удаления задачи и подзадач из списка.

Raspredt – метод разбиение на подзадачи и добавления их в список.

Client – это класс подключенного клиента, он содержит поля сокета, буфера и GUID.

DataTask – это класс отдельного открывающегося окна с информацией задачи и списком подзадач, содержит метод формирования информации и таблицы на форму.

GetIP – это класс получение локального адреса машины.

MyQueue – это класс очереди ожидающих задание клиентов.

Outfile – это класс формирования таблицы Excel и выгрузка ее на машину.

Queuetask – это класс очереди работающих задач.

Queuezadach – это класс очереди подзадач работающей задачи.

Qz – это класс с полями для очереди подзадач.

Основные методы и поля классов очередей:

Elements – создание списка.

Push_Back – метод добавление в начало очереди

Pop_Front – метод извлекает и удаляет первый элемент очереди

Del – метод удаляет элемент по индексу

Size – метод возвращает размер очереди

Clear – метод очисти очереди

У класса MyQueue есть дополнительные методы:

Delete – метод ищет и удаляет нужного клиента

У класса Queuezadach есть дополнительное поле:

Qz – поле с данными о задаче.

Raspredtask – это класс со свойствами и полями для подзадач.

Spisoktask – это класс со свойствами и полями для задач.

6.4 Спецификация клиента

Клиентское приложение так же, как и серверное разработано с использованием графического интерфейса Windows Forms.

Form1 – это основной класс клиента.

Основные методы и поля класса Form1:

workHash – поле для запуска программы перебора пароля hashcat.

Hashcatgo – нить в которой запускается программа hashcat.

UpdateControls – метод обновления формы при подключении и отключении от сервера.

WaitForData – метод ожидания данных от сервера.

OnDataReceived – метод обратного вызова, вызываемая при обнаружении сокетом данных в потоке.

Send – метод отправления фрагмента обратно на сервер.

Closeproc – метод закрытия процесса hashcat.

WorkHash – это класс, который взаимодействует с программой перебора пароля hashcat.

Основные методы и поля класса WorkHash:

hashcatGo – метод запуска и взаимодействия с hashcat.

GetForm – метод взаимодействия с формой из нити.

SocketPacket – это класс со свойствами и полями для подключения.

6.5 Спецификация библиотеки классов

DatainClient – это класс из библиотеки классов со свойствами и полями для сериализации и десериализации данных.

DatainServer – это класс из библиотеки классов со свойствами полями для сериализации и десериализации данных.

7 Тестирование в одиночном и распределенном режимах

После сборки и отладки серверной и клиентской частей, было проведено тестирование распределённой системы, в ходе которого были найдены ее сильные и слабые стороны, намечены дальнейшие пути возможной модернизации.

Также, получили результаты производительности, которые позволили проанализировать перспективу распределённой вычислительной системы подобного рода и дали возможность получить заметный рост производительности, а также обеспечить удобное управление распределением подзадач.

7.1 Результат тестирования в одиночном режиме

В нашем случае был произведен запуск системы в одиночном режиме на компьютере с процессором Intel(R) Core (TM) i5-4570 и видеокартой NVIDIA GeForce GTX 770.

В качестве тестового задания был выбран hccapx файл, содержащий цифровой пароль WPA2, состоящий из 8 символов.

Решение было найдено за 24 минуты 31 секунду (см. рисунок 24)

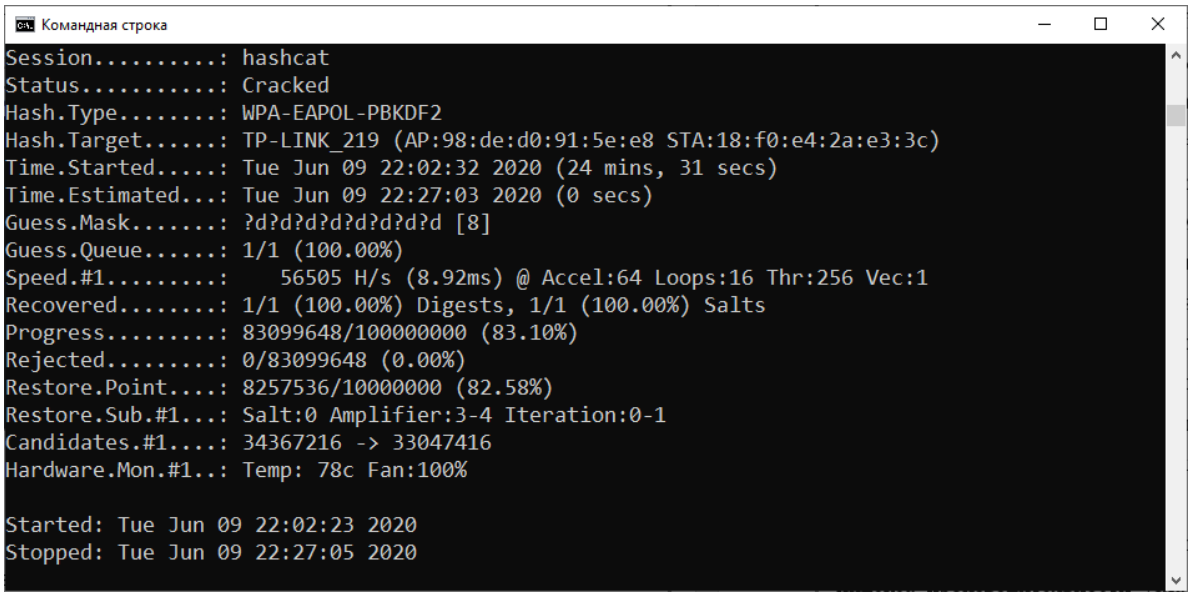


Рисунок 27 - Результат поиска пароля на NVIDIA GeForce GTX 770

Таким образом, тестирование в одиночном режиме показало, что программа hashcat запускается как положено и находит результат за достаточно большое время.

7.2 Результат тестирования в распределенном режиме

В распределенном режиме система работает по основному своему назначению — увеличить производительность с использованием распределенных вычислений на большом количестве узлов.

После того, как система была развернута на 4 компьютерах для выполнения распределенного поиска, был проведен тестовый запуск распределение подзадач

Конфигурация компьютеров, входящих в систему, следующая:

Intel(R) Core (TM) i5-4570 / NVIDIA GeForce GTX 770 – 1 шт

Intel(R) Core (TM) i5-8250U / NVIDIA GeForce MX 150 – 1 шт

Intel(R) Core (TM) i5-6300HQ / NVIDIA GeForce GTX 960M – 1 шт

AMD Ryzen™ 5 2600 / Radeon™ RX 570 – 1 шт

Для расчета также, как и в предыдущем случае был использован файл с хэш-сверткой пароля WPA2, состоящего из 8 цифровых символов.

В распределённом режиме диапазон поиска был поделен на 10 равных частей, каждая из которых выдавалась вычислительному узлу без привязки. То есть первый освободившийся клиент получал задачу из очереди подзадач. Время поиска пароля продемонстрированы при выдачи подзадачи , при которой клиент найдет пароль на ту же вычислительную машину (Intel(R) Core (TM) i5-4570 / NVIDIA GeForce GTX 770), что и при тесте в одиночном режиме (таблица 1, гистограмма 26).

Таблица 1 - время расшифровки

№	Режим расшифровки	Время расчета
1	Одиночный	24 минуты 31 секунд
2	Распределенный	3 минуты 30 секунд

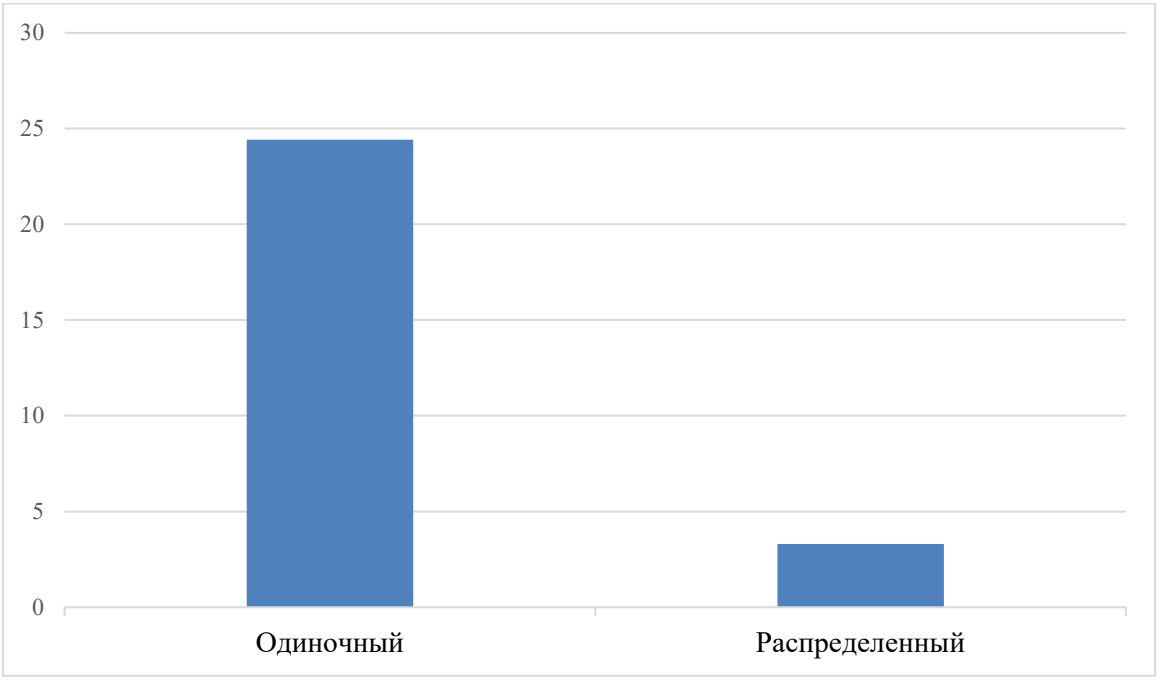


Рисунок 28 - гистограмма сравнения времени расшифровки

Так же было произведено тестирование полного перебора всех подзадач, при котором в обоих режимах пароль не будет найден. То есть расчетное время прохождение по всем значениям маски. (таблица 2, гистограмма 27).

Таблица 2 - время полного прохождения по хэш-значению

№	Режим расшифровки	Время расчета
1	Одиночный	29 минуты 54 секунд
2	Распределенный	12 минуты 16 секунд

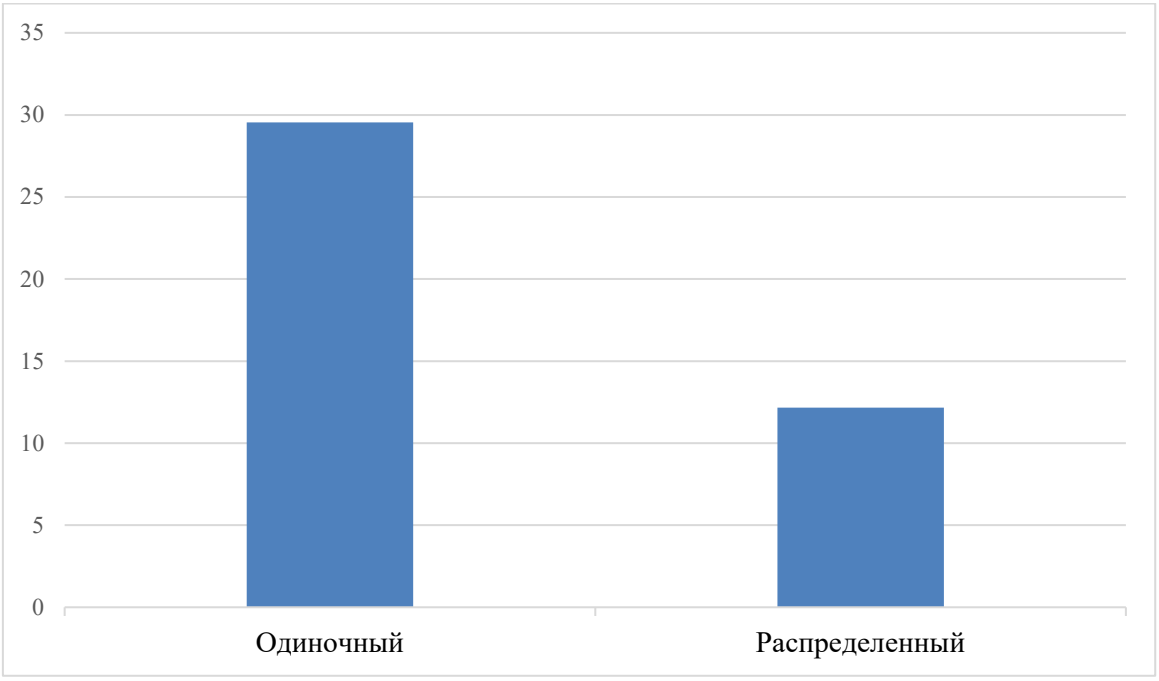


Рисунок 29 - гистограмма сравнения времени полного прохождения по хэш-значению

Из результатов тестирования можно сделать вывод, что с клиент-серверным приложеним пользователь удобнее из-за присутсвия графического интерфейса. Так же время перебора в первом случае, при удачно выданной подзадачи клиенту, время расшифровки в распределнном режиме занимает 3 минуты 30 секунд, что в 8 раз быстрее, чем расшифровка в одниночном режиме. Во втором же случае, при тестировании расчетого времени полного прохождения по всем подзадачам (маскам) время уменьшилось в 2,5 раза.

8 Безопасность жизнедеятельности

8.1 Общие требования безопасности

К работе на персональном компьютере допускаются лица, прошедшие обучение безопасным методам труда, вводный инструктаж, первичный инструктаж на рабочем месте.

При эксплуатации персонального компьютера на работника могут оказывать действие следующие опасные и вредные производственные факторы:

- повышенный уровень электромагнитных излучений;
- повышенный уровень статического электричества;
- пониженная ионизация воздуха;
- статические физические перегрузки;
- перенапряжение зрительных анализаторов.

Рабочие места с компьютерами должны размещаться таким образом, чтобы расстояние от экрана одного видеомонитора до тыла другого было не менее 2,0 м, а расстояние между боковыми поверхностями видеомониторов - не менее 1,2 м.

Рабочие места с персональными компьютерами по отношению к световым проемам должны располагаться так, чтобы естественный свет падал сбоку, преимущественно слева.

Оконные проемы в помещениях, где используются персональные компьютеры, должны быть оборудованы регулируемыми устройствами типа: жалюзи, занавесей, внешних козырьков и др.

Высота рабочей поверхности стола должна регулироваться в пределах 680 - 800 мм; при отсутствии такой возможности высота рабочей поверхности стола должна составлять 725 мм.

Рабочий стол должен иметь пространство для ног высотой не менее 600 мм, глубиной на уровне колен не менее 450 мм и на уровне вытянутых ног не менее 650 мм.

Рабочий стул (кресло) должен быть подъемно - поворотным и регулируемым по высоте и углам наклона сиденья и спинки, а также - расстоянию спинки от переднего края сиденья.

Рабочее место должно быть оборудовано подставкой для ног, имеющей ширину не менее 300 мм, глубину не менее 400 мм, регулировку по высоте в пределах до 150 мм и по углу наклона опорной поверхности подставки до 20 градусов; поверхность подставки должна быть рифленой и иметь по переднему краю бортик высотой 10 мм.

Рабочее место с персональным компьютером должно быть оснащено легко перемещаемым пюпитром для документов.

8.2 Требования безопасности перед началом работы

Отрегулировать освещение на рабочем месте, убедиться в отсутствии бликов на экране.

Убедиться в наличии заземления системного блока, монитора и защитного экрана.

Протереть антистатической салфеткой поверхность экрана монитора и защитного экрана.

Проверить правильность установки стола, стула, подставки для ног, пюпитра, угла наклона экрана, положение клавиатуры, положение "мыши" на специальном коврике, при необходимости произвести регулировку рабочего стола и кресла, а также расположение элементов компьютера в соответствии с требованиями эргономики и в целях исключения неудобных поз и длительных напряжений тела.

Заключение

В ходе выполнения дипломного проекта было разработано клиент-серверное приложение для распределенной расшифровки сверток паролей. Разработанная система представляет собой распределенную вычислительную систему, предназначенную для создания и управления задачами для распределенной обработки, планирования вычислений в распределенной сети вычислительных узлов, выдачи подзадач и сбора результатов.

В качестве задачи, реализованной в системе, выбрана задача восстановления пароля по его хэш значению.

В результате выполнения работы достигнуто следующее:

- разработана модель функционирования системы, разработаны алгоритмы и реализующие их программы для серверных и вычислительных узлов;

- разработан протокол передачи данных узлам и возврата результатов от узлов на сервер с помощью протокола TCP;

- осуществлено тестирование работы созданной распределенной системы на в нескольких режимах.

Производительность и масштабируемость системы соответствует ожидаемой для кластерной распределенной системы рассматриваемого вида.

В процессе тестирования приложения были выявлены различные проблемы. При отключении клиента, который работал над решением подзадачи, подзадача не возвращалась обратно в очередь. Данная проблема была устранена путем создания для каждого клиента своего GUID-идентификатора и при получении подзадачи, сервер записывал идентификатор в хэш-таблицу подзадачи. При отключении клиента приложение проверяет его GUID в хэш-таблице подзадачи, если идентификатор найден, подзадача возвращается в очередь. Так же одна из ошибок была, возврат клиента в очередь ожидающих. Ошибка была устранена с помощью захвата блокировки для доступа к разделяемым ресурсам. Таким же образом, для исключения ошибок добавления задачи, было реализовано добавление, удаление и распределение на подзадачи.

В дальнейшем приложение может быть улучшено путем добавления неравномерного распределения с учетом производительности узлов и пропускной способности каналов. Реализация выбор хэш-значения для перебора так же увеличит гибкость приложения. Также не были рассмотрены вопросы как обеспечение надежности и отказоустойчивости.

Список использованных источников

1. HackWare.ru - Этичный хакинг и тестирование на проникновение, информационная безопасность: https://hackware.ru/
2. А.Н. Токарев, А.В. Прошкин Методика расчета объемов подзадач при равномерном делении в распределенных вычислительных системах с ненадежными узлами. НИТИС-2017, г.Пенза, 2017г.
3. Олифер Виктор, Олифер Наталья Компьютерные сети. Принципы, технологии, протоколы: Юбилейное издание. — СПб.: Питер, 2020. — 1008 с.: ил. — (Серия «Учебник для вузов»)
4. Эндрюс Г.Р. Основы многопоточного, параллельного и распределенного программирования. Пер.с англ. — М.: Издательский дом «Вильямс», 2003.
5. Воеводин В.В., Воеводин Вл.В. Параллельные вычисления. — СПб.: БХВ-Петербург, 2002.
6. "Консультант Плюс" - законодательство РФ: кодексы, законы, указы, постановления Правительства Российской Федерации, нормативные акты: https://www.consultant.ru/

Приложение А Листинги программы сервера

Приложение А.1 Файл «Form1.cs»

```
using System;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using LibraryData;
using System.Text;
using System.Linq;
using System.Threading;

namespace DiplomRab
{
    public partial class Form1 : Form
    {
        private Socket server;
        myQueue myqueue = new myQueue();
        Queuetask queuetask = new Queuetask();
        Queuezadach queuezadach = new Queuezadach();
        MemoryStream s;
        bool istheretask = false;
        bool taskdel = false;
        private const int max = 100;
        addtask addtask = new addtask();
        Thread Deltask, sraspred;
        private Client[] clients = new Client[max];
        private int Count = 0;
        private    static    ManualResetEvent    mre    =    new
ManualResetEvent(true);
        private object con_dis = new object();
        public Form1()
        {
            InitializeComponent();
            var Descript = this.Handle;
            textBox2.Text = GetIP.getIP();
            label4.Text = Count.ToString();
            comboBox1.SelectedIndex = 0;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            Count = 0;
            try
            {
                if (textBox3.Text == "")
                {
                    MessageBox.Show("Введите порт для подключения
клиентов.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
                    return;
                }
                string portStr = textBox3.Text;
                int port = Convert.ToInt32(portStr);
                server  =  new  Socket(AddressFamily.InterNetwork,
SocketType.Stream, ProtocolType.Tcp);
                IPEndPoint ipLocal = new IPEndPoint(IPAddress.Any,
port);
                server.Bind(ipLocal);
                server.Listen(10);
                server.BeginAccept(new
AsyncCallback(OnClientConnect), null);
                UpdateControls(true);
                richTextBox1.AppendText("Сервер работает\n");
            }
            catch (SocketException)
            {
                MessageBox.Show("Ошибка сервера");
            }
            catch (FormatException)
            {
                MessageBox.Show("Данные некорректны");
            }
        }
        private void button2_Click(object sender, EventArgs e)
        {
            queuezadach.Clear();
            queuetask.Clear();
            myqueue.Clear();
            addtask.listraspred.Clear();
            addtask.listtask.Clear();
            CloseSockets();
            UpdateControls(false);
            richTextBox1.AppendText("Сервер выключен\n");
        }
        private void button3_Click(object sender, EventArgs e)
        {
            OpenFileDialog openFileDialog1 = new OpenFileDialog() {
Filter = "Файл захвата пакетов(*.hccapx)|*.hccapx" };
            if (openFileDialog1.ShowDialog() == DialogResult.OK)
                textBox1.Text = openFileDialog1.FileName;
        }
        private void button4_Click(object sender, EventArgs e)
        {
            if (listBox1.SelectedIndex < 0)
            {
                MessageBox.Show(
                       "Выберете задачу",
                       "Ошибка",
                       MessageBoxButtons.OK,
                       MessageBoxIcon.Error);
                return;
            }
```

```
            int indx = listBox1.SelectedIndex;
            string name = listBox1.SelectedItem.ToString();
            Deltask = new Thread(delegate () { deltask(indx, name);
});
            Deltask.IsBackground = true;
            Deltask.Start();
        }
        private void button5_Click(object sender, EventArgs e)
        {
            AboutBox1 about = new AboutBox1();
            about.Show();
        }
        private void button6_Click(object sender, EventArgs e)
        {
            gotask();
        }
        private void button7_Click(object sender, EventArgs e)
        {
            if (textBox1.Text != "Файл не выбран")
            {
                if   (textBox4.Text.Trim()   !=   String.Empty   &&
textBox4.Text.Length <= 16)
                {
                    if (!addtask.listtask.Any(item => item.nametask
== textBox4.Text.Trim()))
                    {
                        string pathtask = textBox1.Text.ToString();
                        string nametask = textBox4.Text.ToString();
                        int                  lf                  =
int.Parse(comboBox1.SelectedItem.ToString());
                        if (radioButton1.Checked == true)
                        {
                            addtask.addt(pathtask, nametask, lf, 0,
radioButton1.Text);
                        }
                        else if (radioButton2.Checked == true)
                        {
                            addtask.addt(pathtask, nametask, lf, 1,
radioButton2.Text);
                        }
                        else if (radioButton4.Checked == true)
                        {
                            addtask.addt(pathtask, nametask, lf, 2,
radioButton4.Text);
                        }
                        listBox1.Items.Add(nametask);
                        richTextBox1.AppendText("Добавлена  задача:
" + nametask + "\r\n");
                        textBox1.Clear();
                        textBox4.Clear();
                    }
                    else
                    {
```

```
                        MessageBox.Show(
                        "Задача  с таким именем уже в списке",
                        "Ошибка",
                        MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
                    }
                }
                else
                {
                    MessageBox.Show(
                       "Введите имя задачи не более 16 символов",
                       "Ошибка",
                       MessageBoxButtons.OK,
                       MessageBoxIcon.Error);
                }
            }
            else
            {
                MessageBox.Show(
                   "Выберите файл",
                   "Ошибка",
                   MessageBoxButtons.OK,
                   MessageBoxIcon.Error);
            }
        }
        private     void     listBox1_MouseDown(object     sender,
MouseEventArgs e)
        {
            if (MouseButtons.Right == e.Button)
            {
                int y = e.Y / listBox1.ItemHeight;
                if (y < listBox1.Items.Count)
                    listBox1.SelectedIndex = listBox1.TopIndex + y;
                listBox1 = (ListBox)sender;
                int  selectedIndex  =  listBox1.IndexFromPoint(e.X,
e.Y);
                if (-1 == selectedIndex)
                {
                    listBox1.ContextMenu = null;
                }
                else
                {
                    listBox1.ContextMenuStrip = contextMenuStrip1;
                }
            }
        }
        private void listBox1_DoubliClick(object sender, EventArgs
e)
        {
            if (listBox1.SelectedIndex >= 0)
            {
                opentask();
            }
```

```
        }
        public void Send(Client x, byte[] s, int ind, int i)
        {
            try
            {
                lock (clients)
                {
                    var client = x;
                    if (client != null && client.socket.Connected)
                    {
                        addtask.listraspred[ind].ip[i]            =
client.socket.RemoteEndPoint.ToString();
                        client.socket.Send(s);
                    }
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Не удалось отправить");
            }
        }
        public void Send(Client x)
        {
            try
            {
                lock (clients)
                {
                    var client = x;
                    if (client != null && client.socket.Connected)
                    {

client.socket.Send(Encoding.ASCII.GetBytes("Yes"));
                    }
                }
            }
            catch (Exception)
            {
                MessageBox.Show("Не удалось отправить");
            }
        }
        public void sRaspred()
        {
            int i;
            int ind;
            IFormatter formatter = new BinaryFormatter();
            while (queuetask.Size() != 0)
            {
                int indexrpd = queuetask.Pop_Front();
                for      (int      j      =      0;      j      <
addtask.listraspred[indexrpd].atrib.Count(); j++)
                {
```

```
queuezadach.Push_Back(addtask.listraspred[indexrpd].atrib[j],
indexrpd, j);
                }
                addtask.listtask[indexrpd].inqueue = true;
                addtask.listraspred[indexrpd].clients     =     new
Guid[addtask.listraspred[indexrpd].atrib.Count()];
                while (queuezadach.Size() != 0)
                {
                    if (myqueue.Size() != 0)
                    {
                        i = queuezadach.Front().i;
                        ind = queuezadach.Front().index;
                        using     (FileStream     fs     =     new
FileStream(addtask.listtask[ind].pathtask,           FileMode.Open,
FileAccess.Read))
                        {
                            byte[]              bytes              =
File.ReadAllBytes(addtask.listtask[ind].pathtask);
                            fs.Read(bytes,                          0,
Convert.ToInt32(fs.Length));
                            addtask.listraspred[ind].started[i]   =
DateTime.Now.ToString();
                            addtask.listraspred[ind].clients[i]   =
myqueue.Front().g;
                            DataInClient    datainclient    =    new
DataInClient(
                                ind,
                                i,
                                queuezadach.Pop_Front().podz,
                                bytes,
                                bytes.Length);
                            s = new MemoryStream();
                            formatter = new BinaryFormatter();
                            formatter.Serialize(s, datainclient);
                            Send(myqueue.Pop_Front(),  s.ToArray(),
ind, i);
                            fs.Close();
                        }
                        Thread.Sleep(1000);
                    }
                    mre.WaitOne();
                    Thread.Sleep(2000);
                }
                if (taskdel != true)
                {
                    taskdel = false;
                    addtask.listtask[indexrpd].inqueue = false;
                }
            }
            istheretask = false;
        }
        private void OnClientConnect(IAsyncResult asyn)
```

```
        {
            try
            {
                lock (con_dis)
                {
                    clients[Count]                =                new
Client(server.EndAccept(asyn));
                    myqueue.Push_Front(clients[Count]);
                    WaitForData(clients[Count]);
                    string                    ipcl                    =
((IPEndPoint)clients[Count].socket.RemoteEndPoint).Address.ToStrin
g();
                    richTextBox1.Invoke(new    Action(()    =>    {
richTextBox1.AppendText($"Подключился: {ipcl}" + "\r\n"); }));
                    ++Count;
                    label4.Invoke(new Action(() => { label4.Text =
Count.ToString(); }));
                    server.BeginAccept(new
AsyncCallback(OnClientConnect), null);
                }
            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1", "\nФункция
OnClientConnection: Сокет был закрыт!\n");
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        public void WaitForData(Client client)
        {
            try
            {
                client.socket.BeginReceive(client.dataBuffer,    0,
client.dataBuffer.Length,           SocketFlags.None,           new
AsyncCallback(OnDataReceived), client);
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        public void OnDataReceived(IAsyncResult asyn)
        {
            try
            {
                Client client = (Client)asyn.AsyncState;
                client.socket.EndReceive(asyn);
                IFormatter formatter = new BinaryFormatter();
                MemoryStream s = new MemoryStream();
```

```
                s.Write(client.dataBuffer,                          0,
client.dataBuffer.Length);
                s.Seek(0, SeekOrigin.Begin);
                DataInServer            datainserver            =
(DataInServer)formatter.Deserialize(s);
                string result = datainserver.atrib.TrimEnd('\0');
                int index = datainserver.numtask;
                richTextBox1.Invoke(new     Action(()     =>     {
richTextBox1.AppendText("Задача "+index+":"+result + "\r\n"); }));
                if (result != "Not Found")
                {
                    listBox1.Invoke(new     Action(()     =>     {
listBox1.Items[index]  =  addtask.listtask[index].nametask  +  "✔";
}));

addtask.listraspred[index].result[datainserver.numzadach] = result;
                    queuezadach.Clear();
                }
                else
                {

addtask.listraspred[index].result[datainserver.numzadach]   =   "Not
Found";
                    if
(addtask.listraspred[index].numpod[addtask.listraspred[index].nump
od.Count() - 1] == datainserver.numzadach)
                    {
                        listBox1.Invoke(new    Action(()    =>    {
listBox1.Items[index]  =  addtask.listtask[index].nametask  +  "✖";
}));
                    }
                }

addtask.listraspred[index].clients[datainserver.numzadach]        =
Guid.Empty;

addtask.listraspred[index].finished[datainserver.numzadach]       =
DateTime.Now.ToString(); ;
                myqueue.Push_Back(client);
                WaitForData(client);
            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1", "\nФункция
OnDataReceived: Сокет был закрыт!\n");
            }
            catch (SocketException)
            {
                lock (con_dis)
                {
                    mre.Reset();
                    myqueue.Delete((Client)asyn.AsyncState);
```

```
                    raspredtask res = null;
                    int i = 0;
                    int j = 0;
                    bool br = false;
                    for (i = 0; i < addtask.listraspred.Count();
i++)
                    {
                        for (j = 0; j <
addtask.listraspred[i].counttask; j++)
                        {
                            if (addtask.listraspred.Find(item =>
item.clients[j] == ((Client)asyn.AsyncState).g) != null)
                            {
                                res = addtask.listraspred.Find(item
=> item.clients[j] == ((Client)asyn.AsyncState).g);
                                br = true;
                                break;
                            }
                        }
                        if (br == true)
                        {
                            br = false;
                            break;
                        }
                    }
                    if (res != null) {
queuezadach.Push_Front(res.atrib[j], res.numtask, res.numpod[j]); }
                    mre.Set();
                    richTextBox1.Invoke(new Action(() => {
richTextBox1.AppendText("Клиент отключился" + "\r\n"); }));
                    --Count;
                    label4.Invoke(new Action(() => { label4.Text =
Count.ToString(); }));
                }
            }
        }
        private void UpdateControls(bool listening)
        {
            button1.Enabled = !listening;
            button2.Enabled = listening;
            string connectStatus = listening ? "Включен" :
"Выключен";
            label10.Text = connectStatus;
        }
        void CloseSockets()
        {
            if (server != null)
            {
                server.Close();
            }
            for (int j = 0; j < Count; j++)
            {
                if (clients[j] != null)
```

```
                {
                    clients[j].socket.Close();
                    clients[j] = null;
                }
            }
            myqueue.Clear();
            queuetask.Clear();
            queuezadach.Clear();
            label4.Invoke(new Action(() => { label4.Text = "0"; }));
        }
        void gotask()
        {
            if (listBox1.SelectedIndex < 0)
            {
                MessageBox.Show(
                       "Выберете задачу",
                       "Ошибка",
                       MessageBoxButtons.OK,
                       MessageBoxIcon.Error);
                return;
            }
            int index = listBox1.SelectedIndex;
            listBox1.Items[index]                                    =
addtask.listtask[index].nametask + " 🕐";
            queuetask.Push_Back(index);
            if (istheretask == false)
            {
                istheretask = true;
                sraspred = new Thread(sRaspred);
                sraspred.IsBackground = true;
                sraspred.Start();
            }
        }
        void deltask(int index, string name)
        {
            if (index < 0)
            {
                MessageBox.Show(
                       "Выберете задачу",
                       "Ошибка",
                       MessageBoxButtons.OK,
                       MessageBoxIcon.Error);
                return;
            }
            taskdel = true;
            mre.Reset();
            if (addtask.listtask[index].inqueue == true)
            {
                int j = 0;
                for        (j        =        0;        j        <
addtask.listraspred[index].counttask; j++)
                {
```

```
                    if   (addtask.listraspred[index].clients[j]   !=
Guid.Empty)
                    {
                        for (int i = 0; i < Count; i++)
                        {
                            if           (clients[i].g           ==
addtask.listraspred[index].clients[j])
                            {
                                Send(clients[i]);
                                myqueue.Push_Back(clients[i]);
                            }
                        }
                    }
                }
                queuezadach.Clear();
            }
            else
            {
                queuetask.Del(index);
                queuezadach.Del(index);
            }
            addtask.delt(index);
            mre.Set();
            listBox1.Invoke(new        Action(()        =>        {
listBox1.Items.RemoveAt(index); }));
            richTextBox1.Invoke(new       Action(()       =>       {
richTextBox1.AppendText($"Удаленна задача: {name}" + "\r\n"); }));
        }
        void opentask()
        {
            DataTask              f              =              new
DataTask(addtask.listtask[listBox1.SelectedIndex],
addtask.listraspred[listBox1.SelectedIndex]);
            f.Show();
        }
        private   void   toolStripMenuItem1_Click(object   sender,
EventArgs e)
        {
            opentask();
        }

        private   void   toolStripMenuItem2_Click(object   sender,
EventArgs e)
        {
            gotask();
        }

        private   void   toolStripMenuItem3_Click(object   sender,
EventArgs e)
        {
            int indx = listBox1.SelectedIndex;
            string name = listBox1.SelectedItem.ToString();
```

```
            Deltask = new Thread(delegate () { deltask(indx, name);
});
            Deltask.IsBackground = true;
            Deltask.Start();
        }
    }
}
```

Приложение А.2 Файл «DataTask.cs»

```
using System.IO;
using System.Windows.Forms;
using DiplomRabClient;

namespace DiplomRab
{
    public partial class DataTask : Form
    {
        string nameinout ="";
        public   DataTask(spisoktask   listtask,       raspredtask
listraspred)
        {
            InitializeComponent();
            label2.Text = listtask.nametask;
            nameinout = listtask.nametask;
            textBox2.Text = Path.GetFileName(listtask.pathtask);
            label8.Text = listtask.lengthpass.ToString();
            textBox1.Text = listtask.pathtask;
            label10.Text = listtask.method;
            for (int i = 0; i < listraspred.atrib.Length; i++)
            {
                int rowNumber = dataGridView1.Rows.Add();
                dataGridView1.Rows[rowNumber].Cells[0].Value      =
listraspred.atrib[i];
                if (listraspred.result != null)
                    dataGridView1.Rows[rowNumber].Cells[1].Value  =
listraspred.result[i];
                if (listraspred.ip != null)
                    dataGridView1.Rows[rowNumber].Cells[2].Value  =
listraspred.ip[i];
                if (listraspred.started != null)
                    dataGridView1.Rows[rowNumber].Cells[3].Value  =
listraspred.started[i];
                if (listraspred.finished != null)
                    dataGridView1.Rows[rowNumber].Cells[4].Value  =
listraspred.finished[i];
            }
        }

        private void button1_Click(object sender, System.EventArgs
e) {
           new Outfile(dataGridView1, nameinout);
        }
```

```
    }
}
```

Приложение А.3 Файл «AboutBox1.cs»

```
using System;
using System.Reflection;
using System.Windows.Forms;

namespace DiplomRab
{
    partial class AboutBox1 : Form
    {
        public AboutBox1()
        {
            InitializeComponent();
            this.Text    =    String.Format("О    программе    {0}",
AssemblyTitle);
        }

        #region Методы доступа к атрибутам сборки
        public string AssemblyTitle
        {
            get
            {
                object[]                attributes                =
Assembly.GetExecutingAssembly().GetCustomAttributes(typeof(Assembl
yTitleAttribute), false);
                if (attributes.Length > 0)
                {
                    AssemblyTitleAttribute     titleAttribute     =
(AssemblyTitleAttribute)attributes[0];
                    if (titleAttribute.Title != "")
                    {
                        return titleAttribute.Title;
                    }
                }
                return
System.IO.Path.GetFileNameWithoutExtension(Assembly.GetExecutingAs
sembly().CodeBase);
            }
        }
        #endregion

        private    void    tableLayoutPanel_Paint(object    sender,
PaintEventArgs e)
        {

        }

        private void okButton_Click(object sender, EventArgs e)
        {
            Close();
```

```
        }
    }
}
```

Приложение А.4 Файл «Client.cs»

```
using System;
using System.Net.Sockets;

namespace DiplomRab
{
    public class Client
    {
        public Client(Socket socket)
        {
            this.socket = socket;
        }
        public Socket socket;
        public byte[] dataBuffer = new byte[50000];
        public Guid g =  Guid.NewGuid();
    }
}
```

Приложение А.5 Файл «addtask.cs»

```
using System.IO;
using System.Collections.Generic;

namespace DiplomRab
{
    public class addtask
    {

        public List<spisoktask> listtask = new List<spisoktask>();
        public     List<raspredtask>     listraspred     =     new
List<raspredtask>();
        private object addlock = new object();
        private object dellock = new object();
        private object raspredlock = new object();
        public void addt(string pathtask1, string name, int lf , int
met,string nmet)
        {
            lock (addlock)
            {
                listtask.Add(new   spisoktask()   {   pathtask   =
pathtask1, nametask = name, lengthpass = lf, method = nmet });
                int ct = listtask.Count - 1;
                raspredt(Path.GetFileName(listtask[ct].pathtask),
listtask[ct].lengthpass, met, ct);
            }
        }
        public void delt(int index)
        {
```

```
            lock (dellock)
            {
                listraspred.RemoveAt(index);
                listtask.RemoveAt(index);
            }
        }
        public void raspredt(string patht, int valuecl, int method,
int index)
        {
            lock (raspredlock)
            {
                string atrib = "-a 3 -m 2500 --outfile=adc.pwd --
outfile-format=2 --restore-disable --potfile-disable " + patht + "
";
                if (method == 0)
                {
                    if (valuecl == 8)
                    {
                        string[] line = new string[10];
                        int[] itask = new int[10];
                        for (int i = 0; i <= 9; i++)
                        {
                            line[i] = atrib + $"{i}?d?d?d?d?d?d?d";
                            itask[i] = i;
                        }
                        listraspred.Add(new raspredtask()
                        {
                            atrib = line,
                            result = new string[10],
                            ip = new string[10],
                            started = new string[10],
                            finished = new string[10],
                            numpod = itask,
                            numtask = index,
                            counttask = 10
                        });
                    }
                    else if (valuecl == 9)
                    {
                        string[] line = new string[100];
                        int[] itask = new int[100];
                        for (int i = 0; i <= 9; i++)
                        {
                            for (int j = 0; j <= 9; j++)
                            {
                                line[i * 10 + j] = atrib +
$"{i}{j}?d?d?d?d?d?d?d";
                                itask[i * 10 + j] = i * 10 + j;
                            }
                        }
                        listraspred.Add(new raspredtask()
                        {
                            atrib = line,
```

```
                                result = new string[100],
                                ip = new string[100],
                                started = new string[100],
                                finished = new string[100],
                                numpod = itask,
                                numtask = index,
                                counttask = 100
                            });
                        }
                        else if (valuecl == 10)
                        {
                            string[] line = new string[1000];
                            int[] itask = new int[1000];
                            for (int i = 0; i <= 9; i++)
                            {
                                for (int j = 0; j <= 9; j++)
                                {
                                    for (int k = 0; k <= 9; k++)
                                    {
                                        line[i * 100 + j * 10 + k] =
atrib + $"{i}{j}{k}?d?d?d?d?d?d?d";
                                        itask[i * 100 + j * 10 + k] = i
* 100 + j * 10 + k;
                                    }
                                }
                            }
                            listraspred.Add(new raspredtask()
                            {
                                atrib = line,
                                result = new string[1000],
                                ip = new string[1000],
                                started = new string[1000],
                                finished = new string[1000],
                                numpod = itask,
                                numtask = index,
                                counttask = 1000
                            });
                        }
                    }
                    else if (method == 1)
                    {
                        string dobinline = "";
                        if (valuecl == 8)
                        {
                            dobinline = "?l?l?l?l?l?l?l?l";
                        }
                        else if (valuecl == 9)
                        {
                            dobinline = "?l?l?l?l?l?l?l?l?l";
                        }
                        else if (valuecl == 10)
                        {
                            dobinline = "?l?l?l?l?l?l?l?l?l?l";
```

```
                    }
                    string letters = "abcdefghijklmnopqrstuvwxyz";
                    string[] line = new string[letters.Length];
                    int[] itask = new int[letters.Length];
                    for (int i = 0; i < letters.Length; i++)
                    {
                        line[i]  =  atrib  +  $"{letters[i]}"  +
dobinline;
                        itask[i] = i;
                    }
                    listraspred.Add(new raspredtask()
                    {
                        atrib = line,
                        result = new string[letters.Length],
                        ip = new string[letters.Length],
                        started = new string[letters.Length],
                        finished = new string[letters.Length],
                        numpod = itask,
                        numtask = index,
                        counttask = letters.Length
                    });
                }
                else if (method == 2)
                {
                    string dobinline = "";
                    if (valuecl == 8)
                    {
                        dobinline = "?h?h?h?h?h?h?h?h";
                    }
                    else if (valuecl == 9)
                    {
                        dobinline = "?h?h?h?h?h?h?h?h?h";
                    }
                    else if (valuecl == 10)
                    {
                        dobinline = "?h?h?h?h?h?h?h?h?h?h";
                    }
                    string                letters                =
"0123456789abcdefghijklmnopqrstuvwxyz";
                    string[] line = new string[letters.Length];
                    int[] itask = new int[letters.Length];
                    for (int i = 0; i < letters.Length; i++)
                    {
                        line[i]  =  atrib  +  $"{letters[i]}"  +
dobinline;
                        itask[i] = i;
                    }
                    listraspred.Add(new raspredtask()
                    {
                        atrib = line,
                        result = new string[letters.Length],
                        ip = new string[letters.Length],
                        started = new string[letters.Length],
```

```
                        finished = new string[letters.Length],
                        numpod = itask,
                        numtask = index,
                        counttask = letters.Length
                    });

                }
            }
        }
    }
}
```

Приложение А.6 Файл «GetIP.cs»

```
using System;
using System.Net;
namespace DiplomRab
{
     public class GetIP
    {
        public static String getIP()
        {
            String strHostName = Dns.GetHostName();
            IPHostEntry                iphostentry                =
Dns.GetHostByName(strHostName);
            String IPStr = "";
            foreach         (IPAddress         ipaddress         in
iphostentry.AddressList)
            {
                IPStr = ipaddress.ToString();
                return IPStr;
            }
            return IPStr;
        }
    }
}
```

Приложение А.7 Файл «myQueue.cs»

```
using System.Collections;

namespace DiplomRab
{
    public class myQueue
    {
        private ArrayList elements;
        private object locker = new object();
        public myQueue()
        {
            elements = new ArrayList();
        }

        public void Push_Back(Client element)
```

```
        {
          lock(locker) {
            elements.Add(element);
          }
        }
        public Client Pop_Back()
        {
            var result = (Client)elements[elements.Count - 1];
            elements.RemoveAt(elements.Count - 1);
            return result;
        }
        public Client Back()
        {
            return (Client)elements[elements.Count - 1];
        }
        public void Push_Front(Client element)
        {
            elements.Insert(0, element);
        }
        public Client Pop_Front()
        {
            var result = (Client)elements[0];
            elements.RemoveAt(0);
            return result;
        }
        public Client Front()
        {
            return (Client)elements[0];
        }
        public void Delete(Client cl)
        {
            for (int i = 0; i < elements.Count; i++)
            {
                if (elements[i] == cl) {
                    elements.RemoveAt(i);
                }
            }
        }
        public int Size()
        {
            return elements.Count;
        }
        public void Clear()
        {
            elements.Clear();
        }
    }
}
```

Приложение А.8 Файл «Outfile.cs»

```
using System.Windows.Forms;
using Excel = Microsoft.Office.Interop.Excel;
```

```
namespace DiplomRabClient
{
    public class Outfile
    {
        public Outfile(DataGridView datainsave,string namefile)
        {
            SaveFileDialog saveFileDialog = new SaveFileDialog();
            saveFileDialog.RestoreDirectory = true;
            saveFileDialog.FileName = namefile;
            saveFileDialog.DefaultExt = "xlsx";
            saveFileDialog.Filter = "Книга Excel|*xlsx|All files
(*.*)|*.*";
            if (saveFileDialog.ShowDialog() == DialogResult.OK)
            {
                string path = saveFileDialog.FileName;

                Excel.Application excelapp = new
Excel.Application();
                Excel.Workbook workbook = excelapp.Workbooks.Add();
                Excel.Worksheet worksheet = workbook.ActiveSheet;
                worksheet.Cells[1, 1] = "Параметры запуска";
                worksheet.Cells[1, 2] = "Результат";
                worksheet.Cells[1, 3] = "IP";
                worksheet.Cells[1, 4] = "Start";
                worksheet.Cells[1, 5] = "Finish";
                int rowExcel = 2;
                for (int i = 0; i < datainsave.Rows.Count; i++)
                {
                    worksheet.Cells[rowExcel, "A"] =
datainsave.Rows[i].Cells["Column1"].Value;
                    worksheet.Cells[rowExcel, "B"] =
datainsave.Rows[i].Cells["Column2"].Value;
                    worksheet.Cells[rowExcel, "C"] =
datainsave.Rows[i].Cells["Column3"].Value;
                    worksheet.Cells[rowExcel, "D"] =
datainsave.Rows[i].Cells["Column4"].Value;
                    worksheet.Cells[rowExcel, "E"] =
datainsave.Rows[i].Cells["Column5"].Value;
                    ++rowExcel;
                }
                excelapp.AlertBeforeOverwriting = false;
                workbook.SaveAs(path);
                excelapp.Quit();
            }
        }
    }
}
```

Приложение А.9 Файл «Queuetask.cs»

```
using System.Collections;
namespace DiplomRab
```

```
{
    public class Queuetask
    {
        private ArrayList elements;

        public Queuetask()
        {
            elements = new ArrayList();
        }
        public void Push_Back(int element)
        {
            elements.Add(element);
        }
        public int Pop_Front()
        {
            int result = (int)elements[0];
            elements.RemoveAt(0);
            return result;
        }
        public void Del(int index)
        {
            elements.Remove(index);
        }
        public int Size()
        {
            return elements.Count;
        }
        public void Clear()
        {
            elements.Clear();
        }
    }
}
```

Приложение А.9 Файл «Queuezadach.cs»

```
using System.Collections;

namespace DiplomRab
{
    public class Queuezadach
    {
        private ArrayList elements;

        public Queuezadach()
        {
            elements = new ArrayList();
        }

        public void Push_Back(string element,int index,int i)
        {
            elements.Add(new qz() {podz = element, index=index,i=i
});
```

```
        }
        public qz Pop_Back()
        {
            qz result = (qz)elements[elements.Count - 1];
            elements.RemoveAt(elements.Count - 1);
            return result;
        }
        public string Back()
        {
            return (string)elements[elements.Count - 1];
        }
        public void Del(int index)
        {
            for (int j = 0; j < Size(); j++){
                if (Front().index == index) {
                    elements.RemoveAt(0);
                }
            }
        }
        public void Push_Front(string element, int index, int i)
        {
            elements.Insert(0, new qz() { podz = element, index =
index, i = i });
        }
        public qz Pop_Front()
        {
            qz result = (qz)elements[0];
            elements.RemoveAt(0);
            return result;
        }
        public qz Front()
        {
            return (qz)elements[0];
        }
        public int Size()
        {
            return elements.Count;
        }
        public void Clear()
        {
            elements.Clear();
        }
    }
    public class qz
    {
        public string podz { get; set; }
        public int index { get; set; }
        public int i { get; set; }
    }
}
```

Приложение А.9 Файл «raspredtask.cs»

```
using System;

namespace DiplomRab
{
    public class raspredtask
    {
        public int numtask { get; set; }
        public int counttask { get; set; }
        public int[] numpod { get; set; }
        public string[] atrib { get; set; }
        public string[] result { get; set; }
        public string[] ip { get; set; }
        public string[] started { get; set; }
        public string[] finished { get; set; }
        public Guid[] clients { get; set; }
    }
}
```

Приложение А.10 Файл «spisoktask.cs»

```
namespace DiplomRab
{
    public class spisoktask
    {
        public string pathtask { get; set; }
        public string nametask { get; set; }
        public int lengthpass { get; set; }
        public string method { get; set; }
        public bool inqueue { get; set; }
    }
}
```

Приложение В Листинги программы клиента

Приложение В.1 Файл «Form1.cs»

```
using System;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Diagnostics;
using System.Threading;
using LibraryData;
using System.Linq;
using System.Text;

namespace DiplomRabClient
{
    public partial class Form1 : Form
    {
        WorkHash workHash = new WorkHash();
        private IAsyncResult result;
        public AsyncCallback callBack;
        public Socket clientSocket;
        Thread hashcatgo;
        byte[] goexit = Encoding.ASCII.GetBytes("Yes");
        public Form1()
        {
            InitializeComponent();
        }
        private      void      Form1_FormClosed(object      sender,
FormClosedEventArgs e)
        {
            closeproc();
            foreach           (Process           proc           in
Process.GetProcessesByName("hashcat64"))
            {
                proc.Kill();
            }
        }
        protected override void OnResize(EventArgs e)
        {
            if (WindowState == FormWindowState.Minimized)
            {
                ShowInTaskbar = false;
                notifyIcon1.Visible = true;
            }
            else
            {
                ShowInTaskbar = true;
                notifyIcon1.Visible = false;
```

```
            }
            base.OnResize(e);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (textBox1.Text == "" || textBox2.Text == "")
            {
                MessageBox.Show("Введите данные для
подключения.\n");
                return;
            }
            try
            {
                UpdateControls(false);
                clientSocket = new
Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
                IPAddress ip = IPAddress.Parse(textBox1.Text);
                int iPortNo = Convert.ToInt16(textBox2.Text);
                IPEndPoint ipEnd = new IPEndPoint(ip, iPortNo);
                clientSocket.Connect(ipEnd);
                if (clientSocket.Connected)
                {
                    UpdateControls(true);
                    WaitForData();
                }
            }
            catch (SocketException)
            {
                MessageBox.Show("Не удалось подключиться к
серверу.\n");
                UpdateControls(false);
            }
            catch (FormatException)
            {
                MessageBox.Show("Данные некорректны");
            }
        }
        public void WaitForData()
        {
            try
            {
                if (callBack == null)
                {
                    callBack = new AsyncCallback(OnDataReceived);
                }
                SocketPacket theSocPkt = new SocketPacket
                {
                    thisSocket = clientSocket
                };
                result =
clientSocket.BeginReceive(theSocPkt.dataBuffer, 0,
```

```
theSocPkt.dataBuffer.Length,      SocketFlags.None,      callBack,
theSocPkt);
            }
            catch (SocketException se)
            {
                MessageBox.Show(se.Message);
            }
        }
        private void OnDataReceived(IAsyncResult asyn)
        {
            try
            {
                SocketPacket               theSockId               =
(SocketPacket)asyn.AsyncState;
                theSockId.thisSocket.EndReceive(asyn);
                if (goexit[1] == theSockId.dataBuffer[1]) {
                    closeproc();
                    foreach         (Process         proc         in
Process.GetProcessesByName("hashcat64"))
                    {
                        proc.Kill();
                    }
                    WaitForData();
                    return;
                }
                IFormatter formatter = new BinaryFormatter();
                MemoryStream s = new MemoryStream();
                s.Write(theSockId.dataBuffer,                      0,
theSockId.dataBuffer.Length);
                s.Seek(0, SeekOrigin.Begin);
                DataInClient             datainclient             =
(DataInClient)formatter.Deserialize(s);
                using     (FileStream     fileStream     =     new
FileStream(Application.StartupPath+   @"\hashcat\"+"diplom.hccapx",
FileMode.Create))
                {
                    for (int i = 0; i < datainclient.sizefile; i++)
                    {

fileStream.WriteByte(datainclient.fileb[i]);
                    }
                    fileStream.Seek(0, SeekOrigin.Begin);
                    for (int i = 0; i < datainclient.sizefile; i++)
                    {
                        if         (datainclient.fileb[i]         !=
fileStream.ReadByte())
                        {
                            clientSocket.Close();
                            clientSocket = null;
                            UpdateControls(false);
                            MessageBox.Show("Не   удалось   создать
файл, переподключитесь к серверу");
                            return;
```

```
                        }
                    }
                }
                hashcatgo = new Thread(delegate () {workHash.hashcatGo(datainclient.atrib,
datainclient.numtask,datainclient.numzadach); });
                hashcatgo.IsBackground = true;
                hashcatgo.Start();
                WaitForData();
            }
            catch (ObjectDisposedException)
            {
                System.Diagnostics.Debugger.Log(0, "1", "\nФункция
OnDataReceived: Сокет был закрыт!\n");
            }
            catch (SocketException)
            {
                closeproc();
                foreach (Process proc in
Process.GetProcessesByName("hashcat64"))
                {
                    proc.Kill();
                }
                Action action = () => label5.Text = "Не подключен";
                if (label5.InvokeRequired)
                    label5.Invoke(action);
                else
                    action();
                UpdateControls(false);
                MessageBox.Show("Сервер разорвал соединение\n");
            }
        }
        public void UpdateControls(bool connected)
        {
            button1.Invoke(new Action(() => { button1.Enabled =
!connected; }));
            button2.Invoke(new Action(() => { button2.Enabled =
connected; }));
            string connectStatus = connected ? "Подключен" : "Не
подключен";
            label5.Invoke(new Action(() => { label5.Text =
connectStatus; }));
        }
        private void button2_Click(object sender, EventArgs e)
        {
            closeproc();
            foreach (Process proc in
Process.GetProcessesByName("hashcat64"))
            {
                proc.Kill();
            }
            clientSocket.Close();
            clientSocket = null;
```

```
            UpdateControls(false);
        }
        public void Send(string line,int nt,int nz)
        {
            IFormatter formatter = new BinaryFormatter();
            DataInServer datainclient = new DataInServer(
                                nt,
                                nz,
                                line);
            MemoryStream s = new MemoryStream();
            formatter = new BinaryFormatter();
            formatter.Serialize(s, datainclient);
            try
            {
                if (clientSocket != null)
                {
                    if (clientSocket.Connected)
                    {
                       clientSocket.Send(s.ToArray());
                    }
                }
            }
            catch (Exception exc)
            {
                MessageBox.Show(exc.Message);
            }
        }
        public void closeproc() {
            if (hashcatgo != null)
                if (hashcatgo.IsAlive)
                    hashcatgo.Abort();
        }
        private   void   развернутьToolStripMenuItem_Click_1(object
sender, EventArgs e)
        {
            WindowState = FormWindowState.Normal;
        }
        private    void    закрытьToolStripMenuItem_Click_1(object
sender, EventArgs e)
        {
            Close();
        }
        private void notifyIcon1_MouseDoubleClick_1(object sender,
MouseEventArgs e)
        {
            if (WindowState == FormWindowState.Minimized)
            {
                WindowState = FormWindowState.Normal;
            }
        }
    }
}
```

Приложение В.2 Файл «SocketPacket.cs»

```
using System.Net.Sockets;

namespace DiplomRabClient
{
    public class SocketPacket
    {
        public Socket thisSocket;
        public byte[] dataBuffer = new byte[50000];
    }
}
```

Приложение В.3 Файл «WorkHash.cs»

```
using System;
using System.Windows.Forms;
using System.IO;
using System.Diagnostics;
using System.Threading;

namespace DiplomRabClient
{
    public class WorkHash
    {
        public void hashcatGo(string arg, int nt,int nz)
        {
            var frm = GetForm<Form1>();
            string root = Application.StartupPath + @"\hashcat";
            File.Delete(root + @"\hashcat.log");
            Thread.Sleep(1000);
            ProcessStartInfo startInfo = new ProcessStartInfo
            {
                WorkingDirectory = root,
                CreateNoWindow = true,
                FileName = root + @"\hashcat64.exe",
                WindowStyle = ProcessWindowStyle.Hidden,
                Arguments = arg,
            };
            Process.Start(startInfo).WaitForExit();
            if (!File.Exists(root + @"\hashcat.log")) {
                foreach (Process proc1 in
Process.GetProcessesByName("hashcat64"))
                {
                    proc1.Kill();
                }
                    frm.clientSocket.Close();
                    frm.clientSocket = null;
                    frm.UpdateControls(false);
                    MessageBox.Show("Ошибка, проверьте требования
hashcat:\n");
                    return;
```

```
            }
            string line;
            if (File.Exists(root + @"\adc.pwd"))
            {
                using (StreamReader sr = new StreamReader(root +
@"\adc.pwd", System.Text.Encoding.Default))
                {
                    line = sr.ReadLine();
                }
                File.Delete(root + @"\adc.pwd");
                File.Delete(root + @"\diplom.hccapx");
            }
            else
            {
                line = "Not Found";
            }
            frm.Send(line,nt,nz);
        }
        public static T GetForm<T>() where T : Form
        {
            return (T)Application.OpenForms[typeof(T).Name];
        }
    }
}
```

Приложение С Листинги библиотеки классов

Приложение С.1 Файл «DataInClient.cs»

```
using System;
namespace LibraryData
{
    [Serializable]
    public class DataInClient
    {
        public string atrib { get; set; }
        public byte[] fileb { get; set; }
        public int sizefile{ get; set; }
        public int numtask { get; set; }
        public int numzadach { get; set; }

        public DataInClient(int nt, int nz,string a, byte[] b, int
s)
        {
            atrib = a;
            fileb = b;
            sizefile = s;
            numtask = nt;
            numzadach = nz;
        }
    }
}
```

Приложение С.2 Файл «DataInServer.cs»

```
using System;

namespace LibraryData
{
    [Serializable]
    public class DataInServer
    {
        public int numtask { get; set; }
        public int numzadach { get; set; }
        public string atrib { get; set; }

        public DataInServer(int nt, int nz, string a)
        {
            numtask = nt;
            numzadach = nz;
            atrib = a;
        }
    }
}
```

Приложение D Результаты работы программы

Приложение D.1 Вид начально экрана клиента

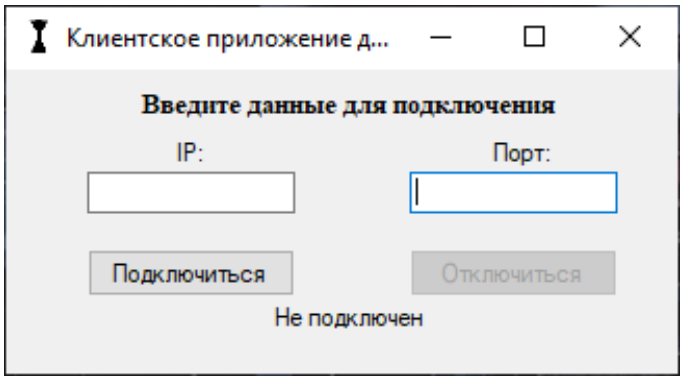


Рисунок 30 - Вид начально экрана клиента

Приложение D.2 Вид свернутого в трей клиента

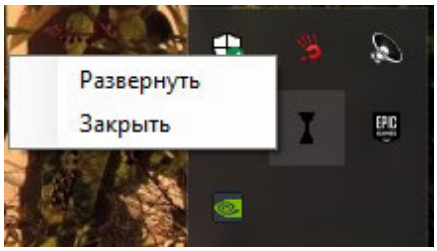


Рисунок 31 - Свернутое в «трей» клиентское приложение

Приложение D.3 Ошибка при подключении к серверу

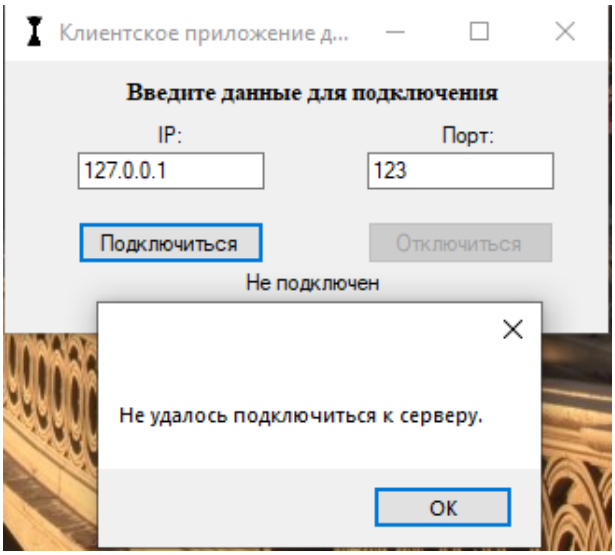


Рисунок 32 - Ошибка подключения к серверу

Приложение D.4 Сообщение об отключении сервера

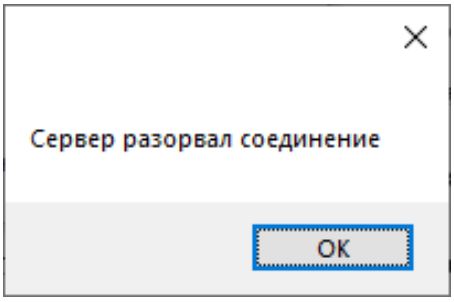


Рисунок 33 - Ошибка отключение сервера

Приложение D.5 Вид начально экрана сервера

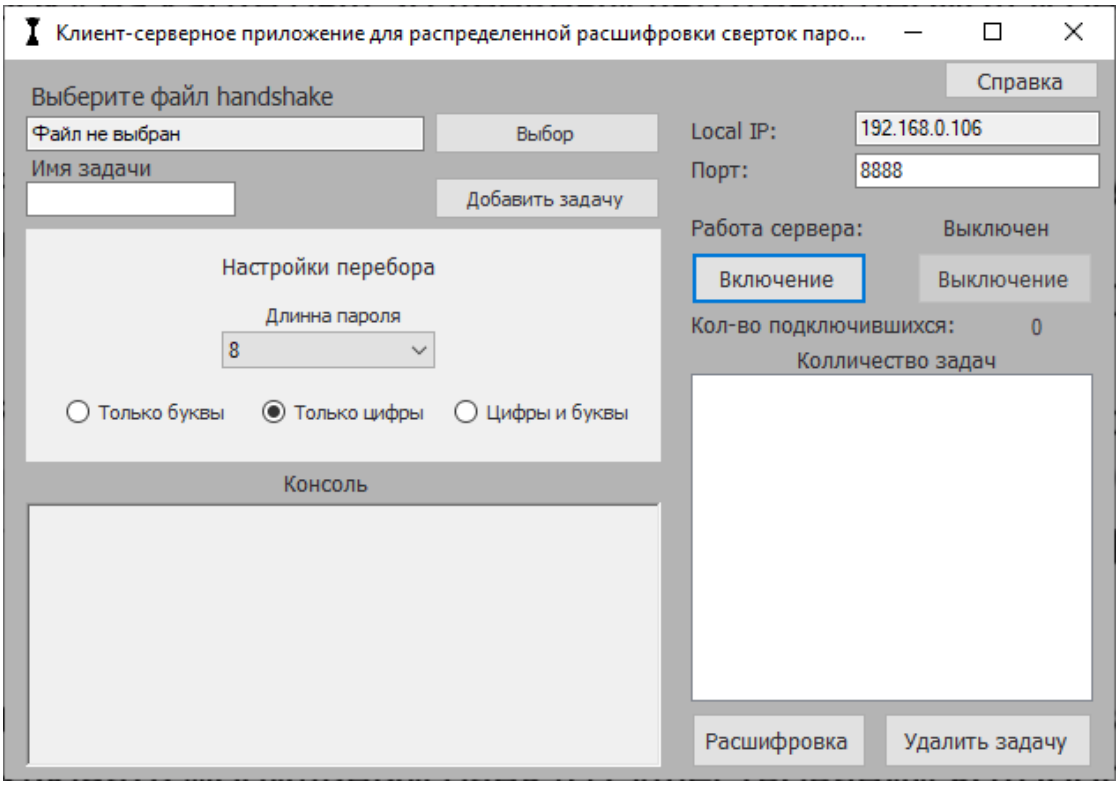


Рисунок 34 - Вид начально экрана сервера

Приложение D.6 Вид экрана с добавленной задачей

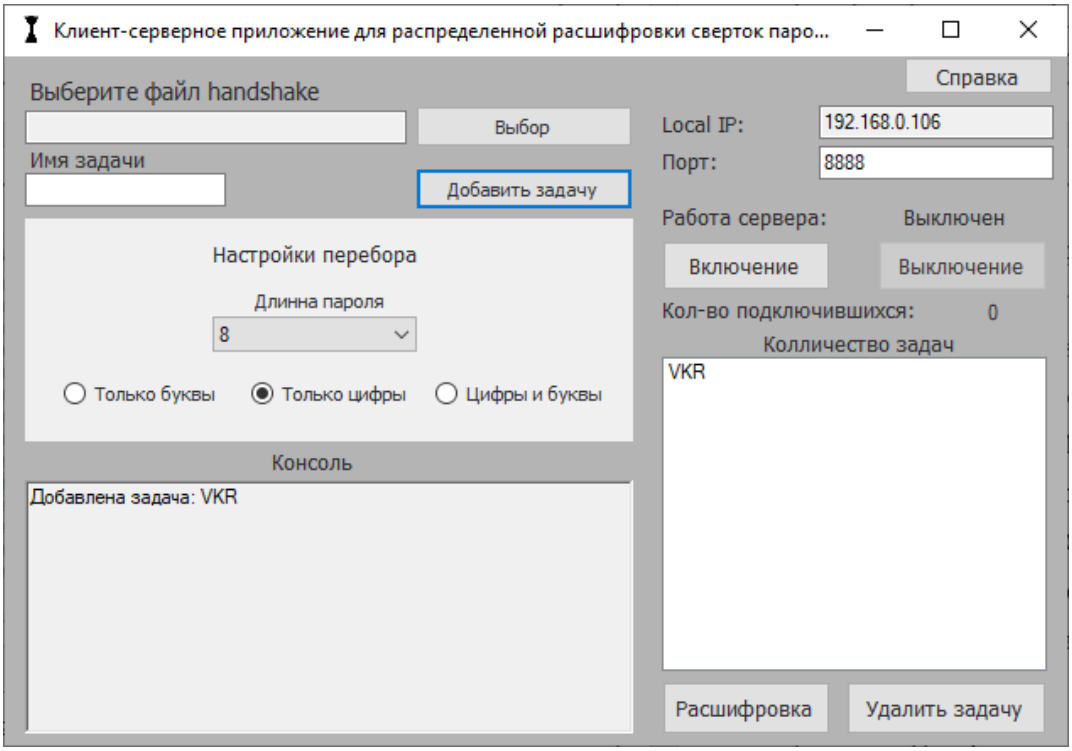


Рисунок 35 - Добавление задачи в список

Приложение D.7 Окно выбора файла handshake

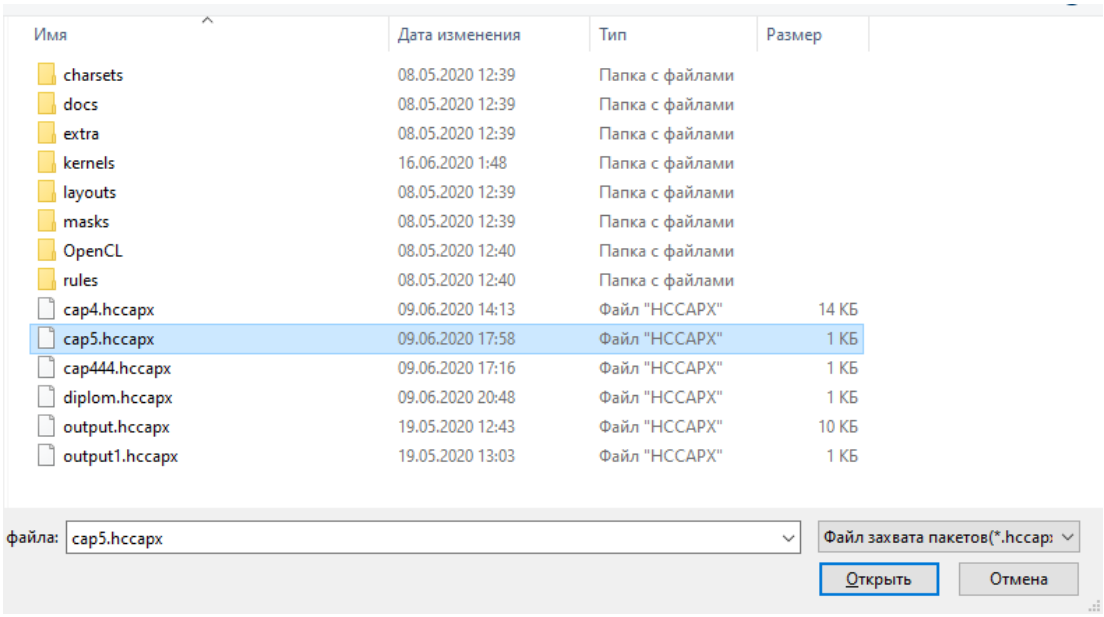


Рисунок 36 - Выбор файла handshake

Приложение D.8 Вид экрана с сообщением об ошибке 1

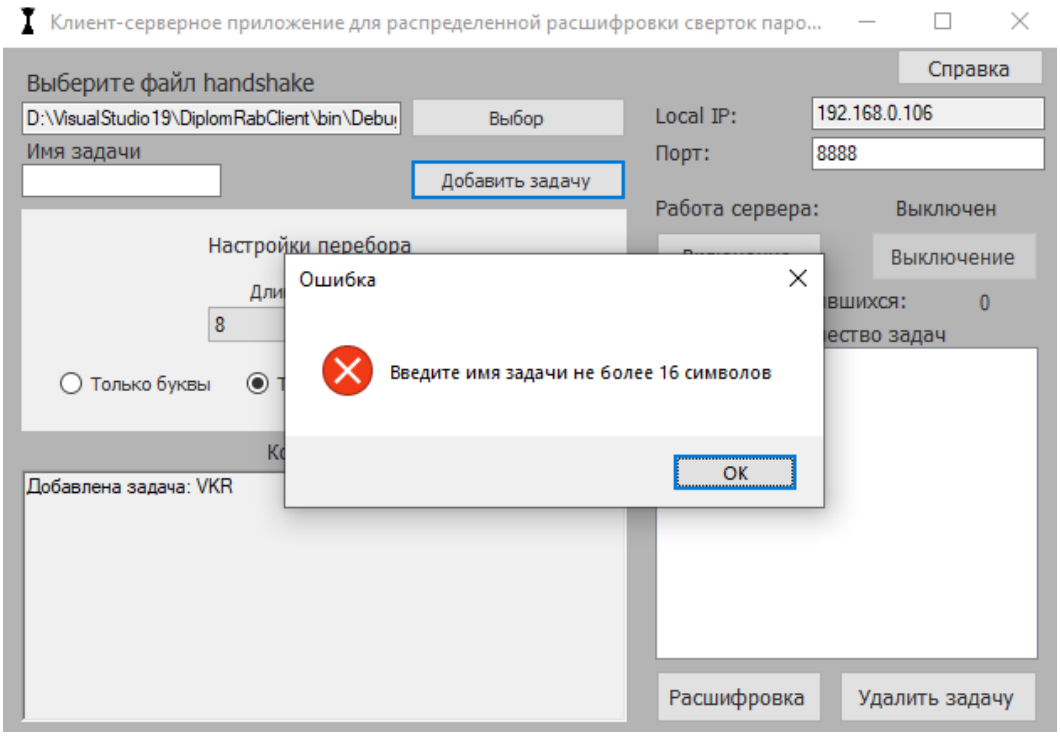


Рисунок 37 - Ошибка при вводе имени задачи

Приложение D.9 Вид экрана с сообщением об ошибке 2

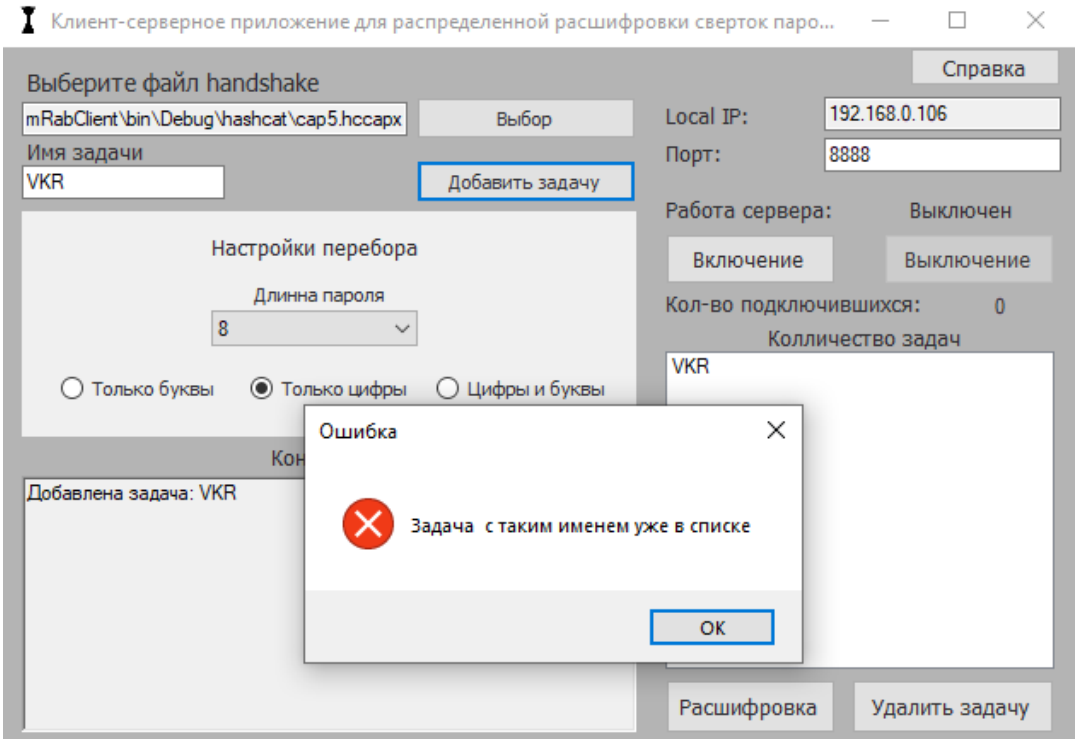


Рисунок 38 - Ошибка при вводе имени задач, которая есть в списке

Приложение D.10 Вид экрана с сообщением об ошибке 3

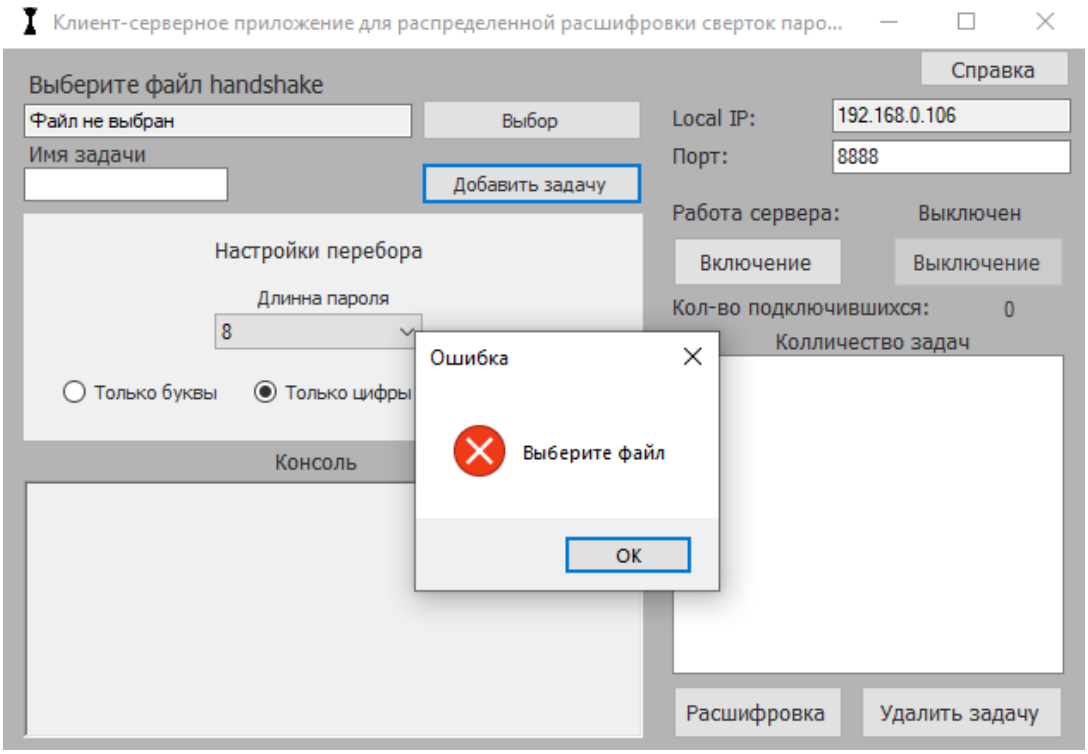


Рисунок 39 - Ошибка, если файл не выбран

Приложение D.11 Контекстное меню

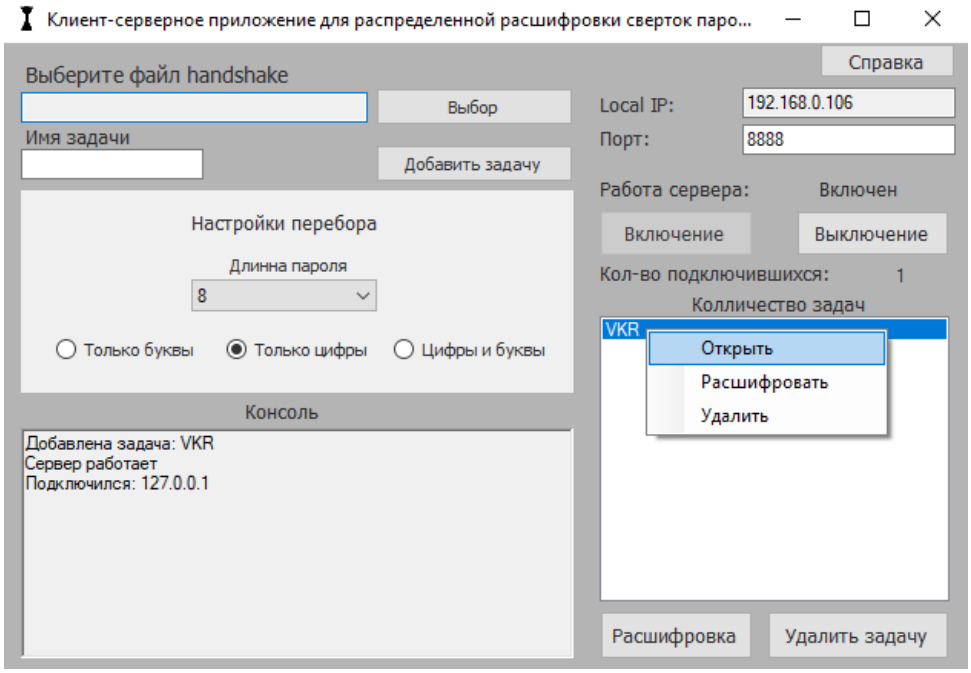


Рисунок 40 - Демонстрация контекстного меню

Приложение D.12 Вид экрана при расшифровке

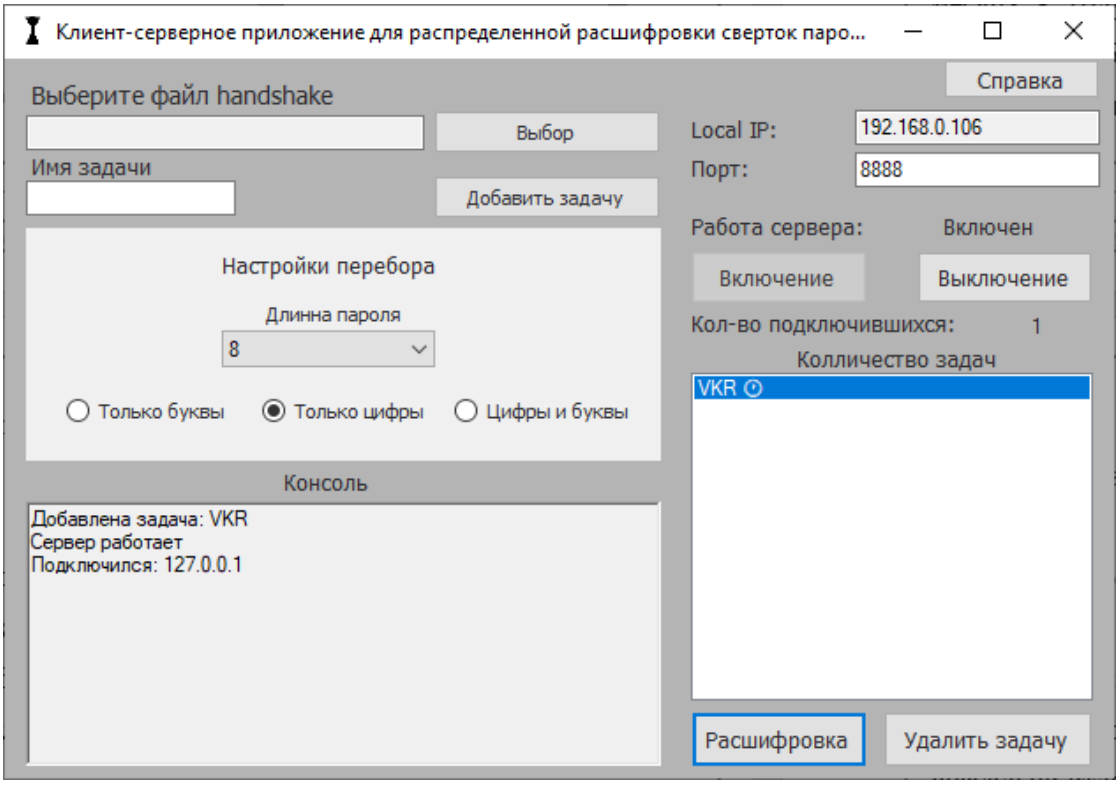


Рисунок 41 - Демонстрация работы расшифровки задачи

Приложение D.13 Вид экрана при удачном поиске пароля

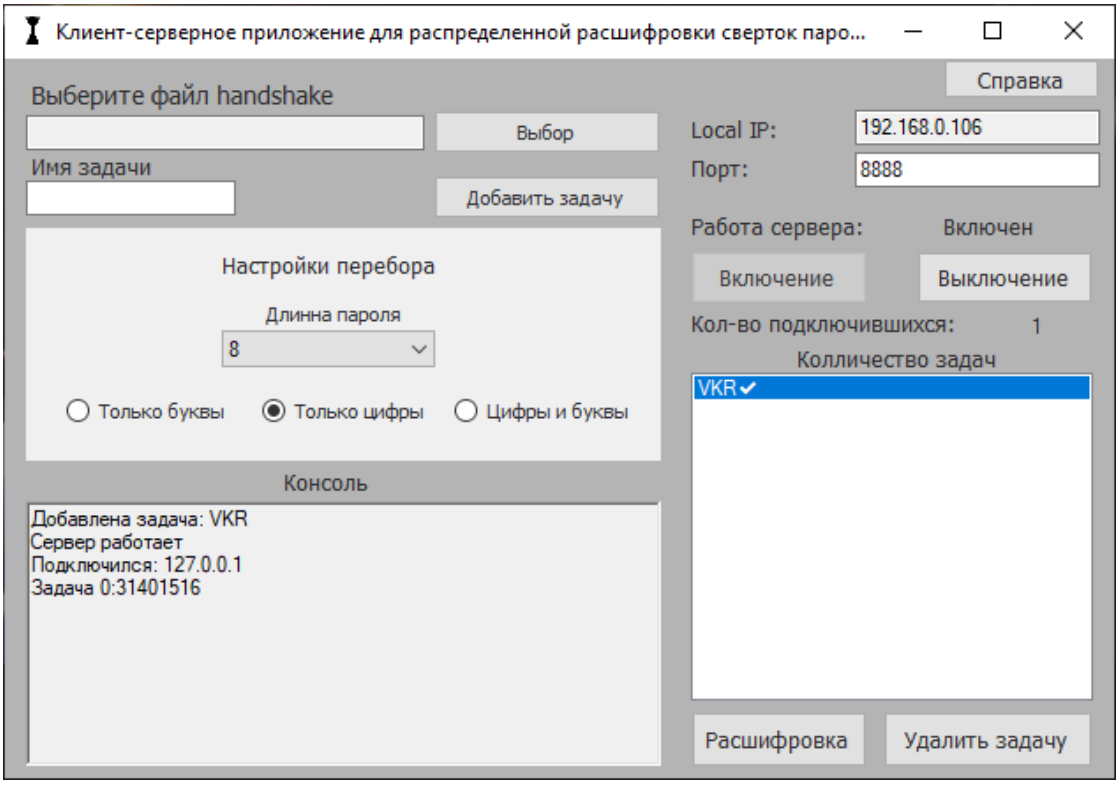


Рисунок 42 - Демонстрация завершенной задаче и удачного поиска пароля

Приложение D.14 Вид экрана работы сервера

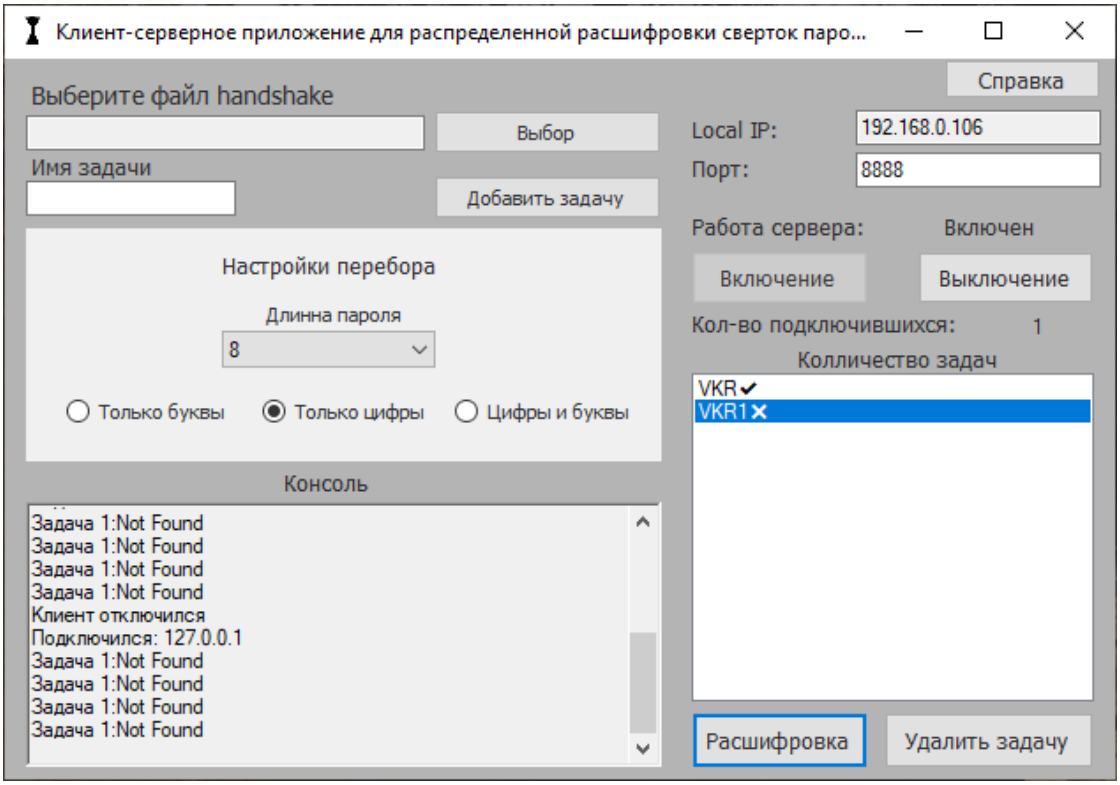


Рисунок 43 - Демонстрация найденного и не найденного пароля

Приложение D.15 Вид экрана при удалении задачи

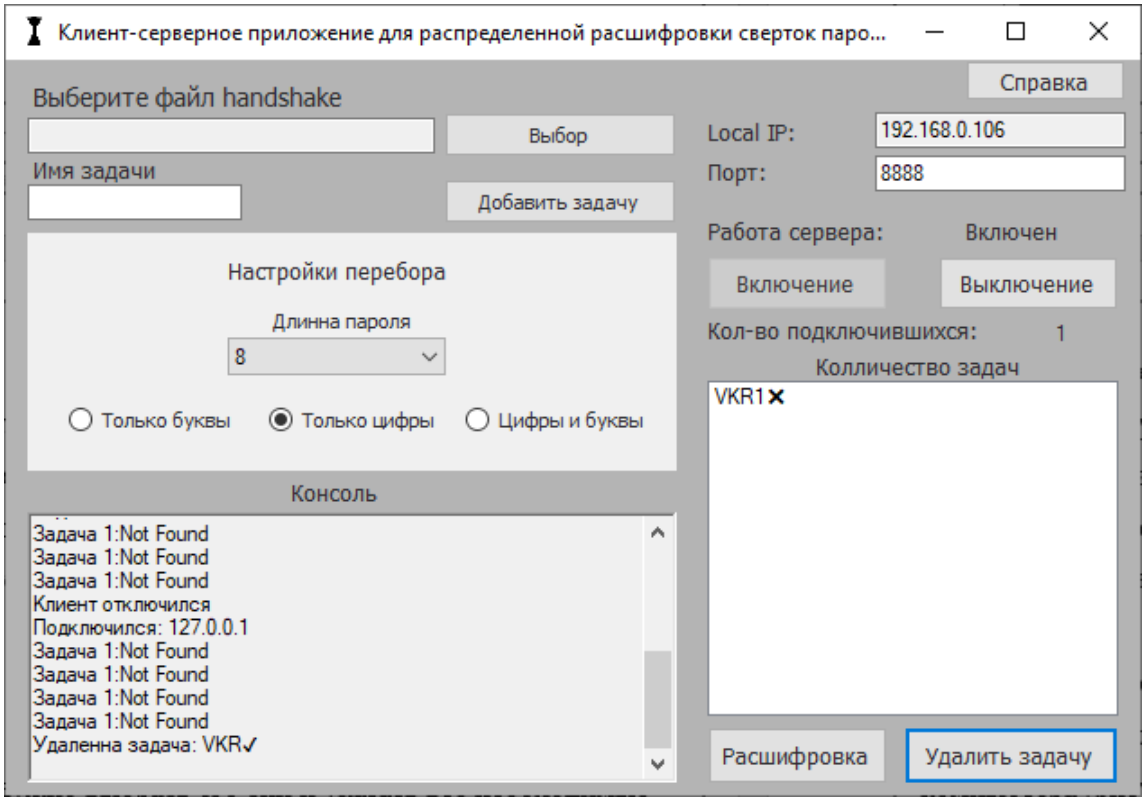


Рисунок 44 - Демонстрация удаление задачи

Приложение D.16 Вид таблицы подзадач

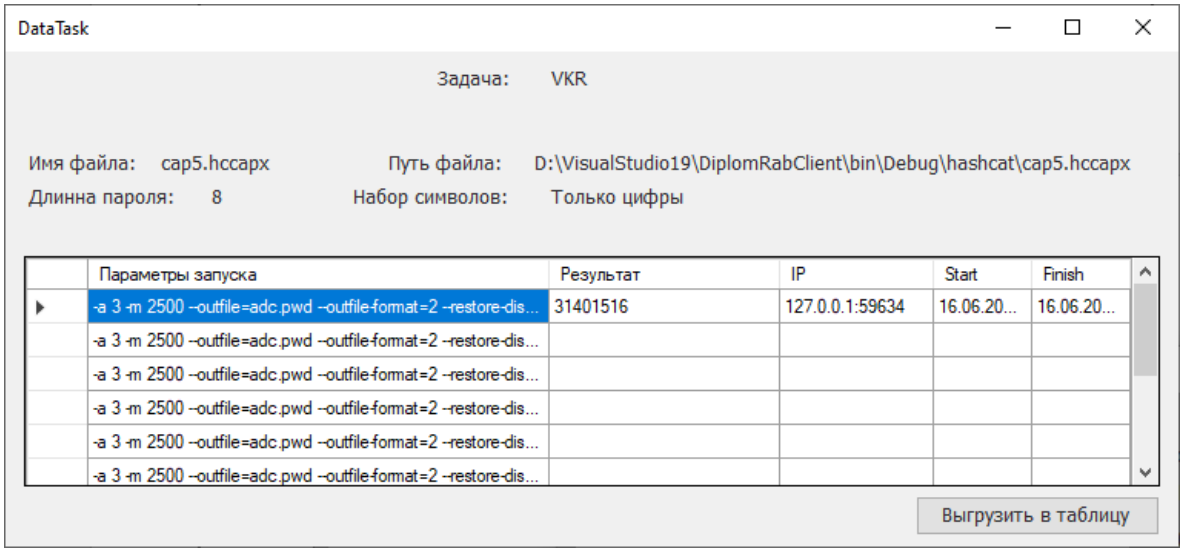


Рисунок 45 - Открытая задача с информацией и подзадачами

Приложение D.17 Вид таблицы подзадач

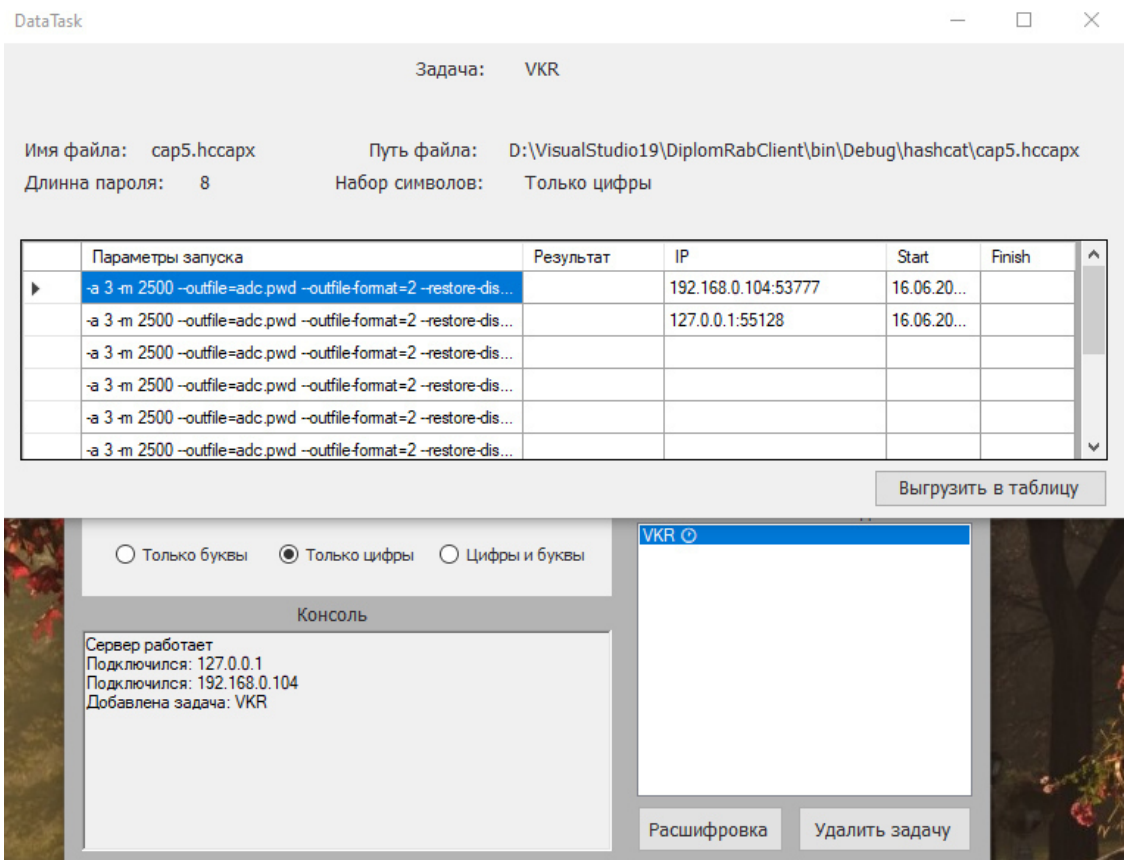


Рисунок 46 - клиенты работающие над подзадачами

Приложение D.18 Вид таблицы подзадач

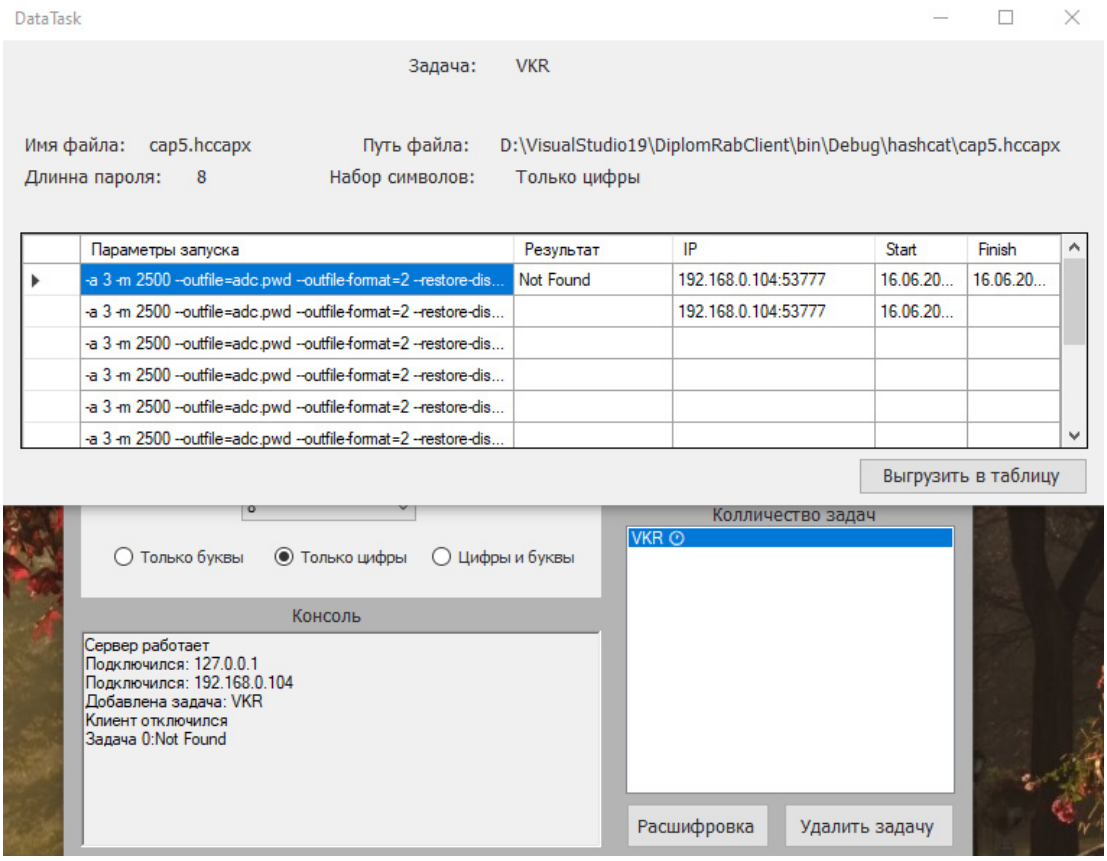


Рисунок 47 - Освободившемуся клиенту выдается подзадача отключенного клиента

Приложение D.19 Выгруженная таблица в Excel

Таблица 3 – таблица подзадач с атрибутами

Параметры запуска	Результат	IP	Start	Finish
-a 3 -m 2500 --outfile=adc.pwd --outfile-format=2 --restore-disable --potfile-disable output.hccapx 90000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:33:10	05.06.2020 1:33:33

Продолжение таблицы 3

-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 80000?d?d?d	Not Found	127.0.0.1:51308	05.06.2020 1:33:13	05.06.2020 1:33:35
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 70000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:33:34	05.06.2020 1:33:48
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 60000?d?d?d	Not Found	127.0.0.1:51308	05.06.2020 1:33:37	05.06.2020 1:33:50
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 50000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:33:48	05.06.2020 1:34:02

Продолжение таблицы 3

-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 40000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:34:17	05.06.2020 1:34:32
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 30000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:34:02	05.06.2020 1:34:16
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 20000?d?d?d	Not Found	127.0.0.1:51321	05.06.2020 1:34:24	05.06.2020 1:34:37
-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 10000?d?d?d	Not Found	192.168.0.104:49304	05.06.2020 1:34:33	05.06.2020 1:34:51

Продолжение таблицы 3

-a 3 -m 2500 -- outfile=adc.pwd -- outfile-format=2 -- restore-disable -- potfile-disable output.hccapx 00000?d?d?d	00000111	127.0.0.1:51321	05.06.2020 1:34:38	05.06.2020 1:34:51

Приложение Е UML-диаграммы приложения

Приложение Е.1 – UML-диаграмма классов сервера

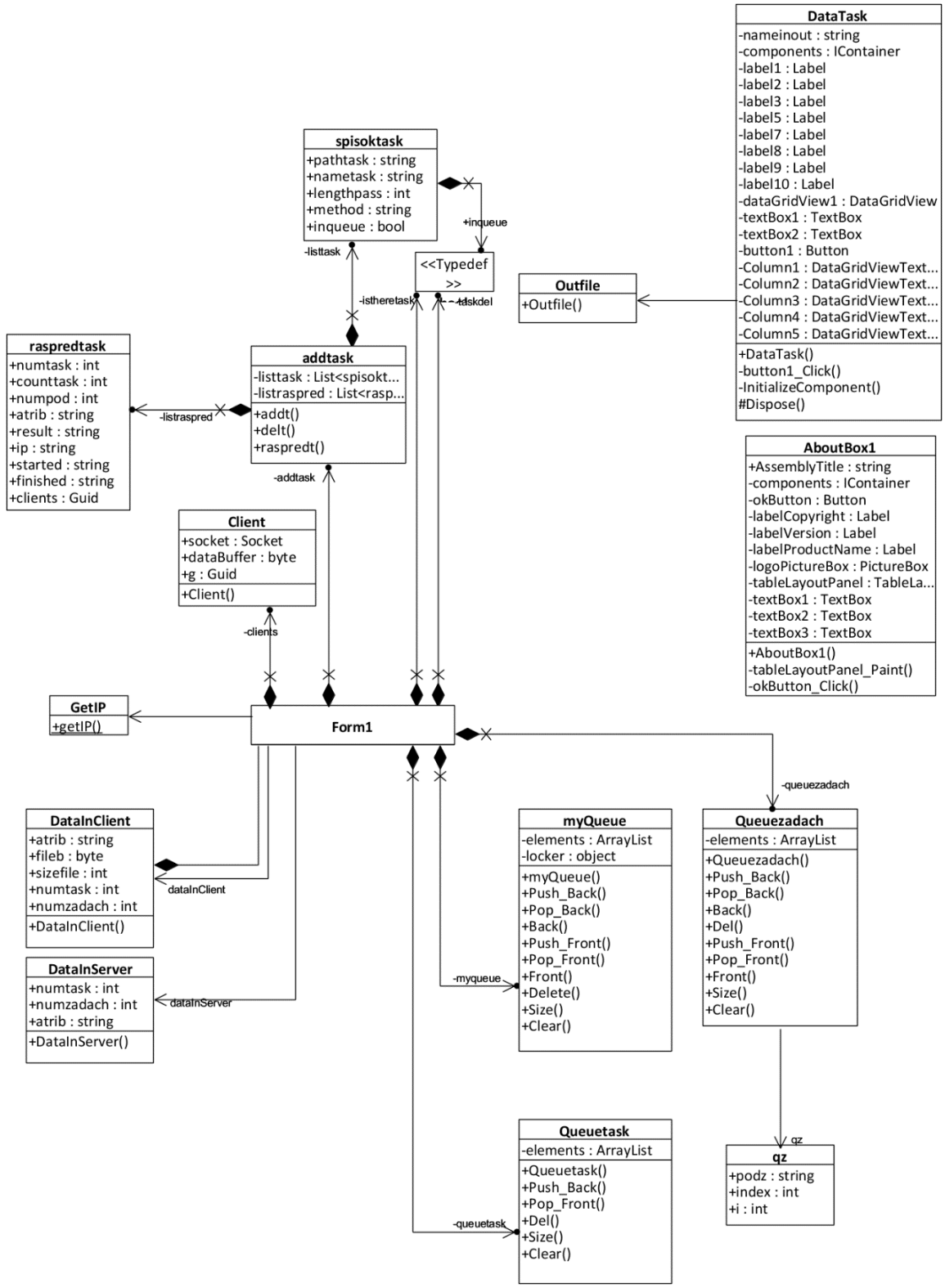


Рисунок 48 - UML-диаграмма классов сервера

Приложение Е.2 – UML-диаграмма классов клиента

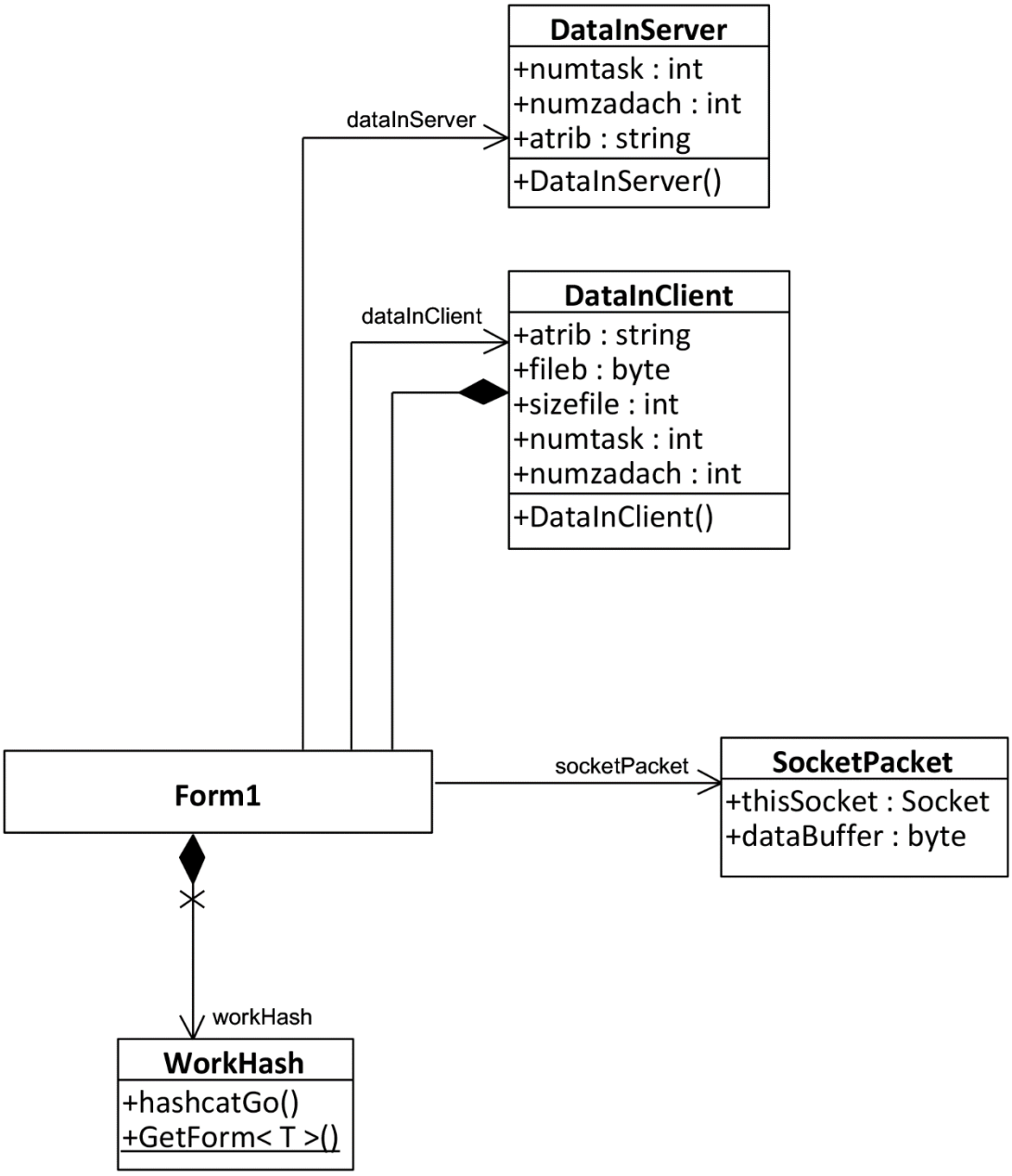


Рисунок 49 - UML-диаграмма классов клиента

Приложение Е.3 – UML-диаграмма вариантов использования сервера



Рисунок 50 - UML-диаграмма вариантов использования сервера

Приложение Е.4 – UML-диаграмма вариантов использования клиента

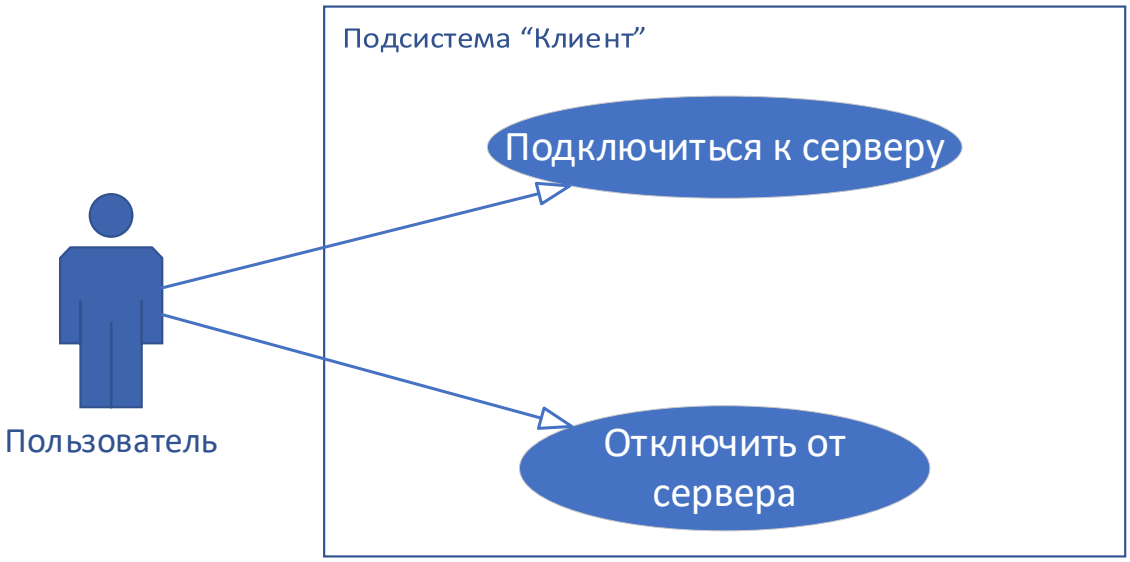


Рисунок 51 - UML-диаграмма вариантов использования клиента

Приложение Е.5 – UML-диаграмма деятельности сервера

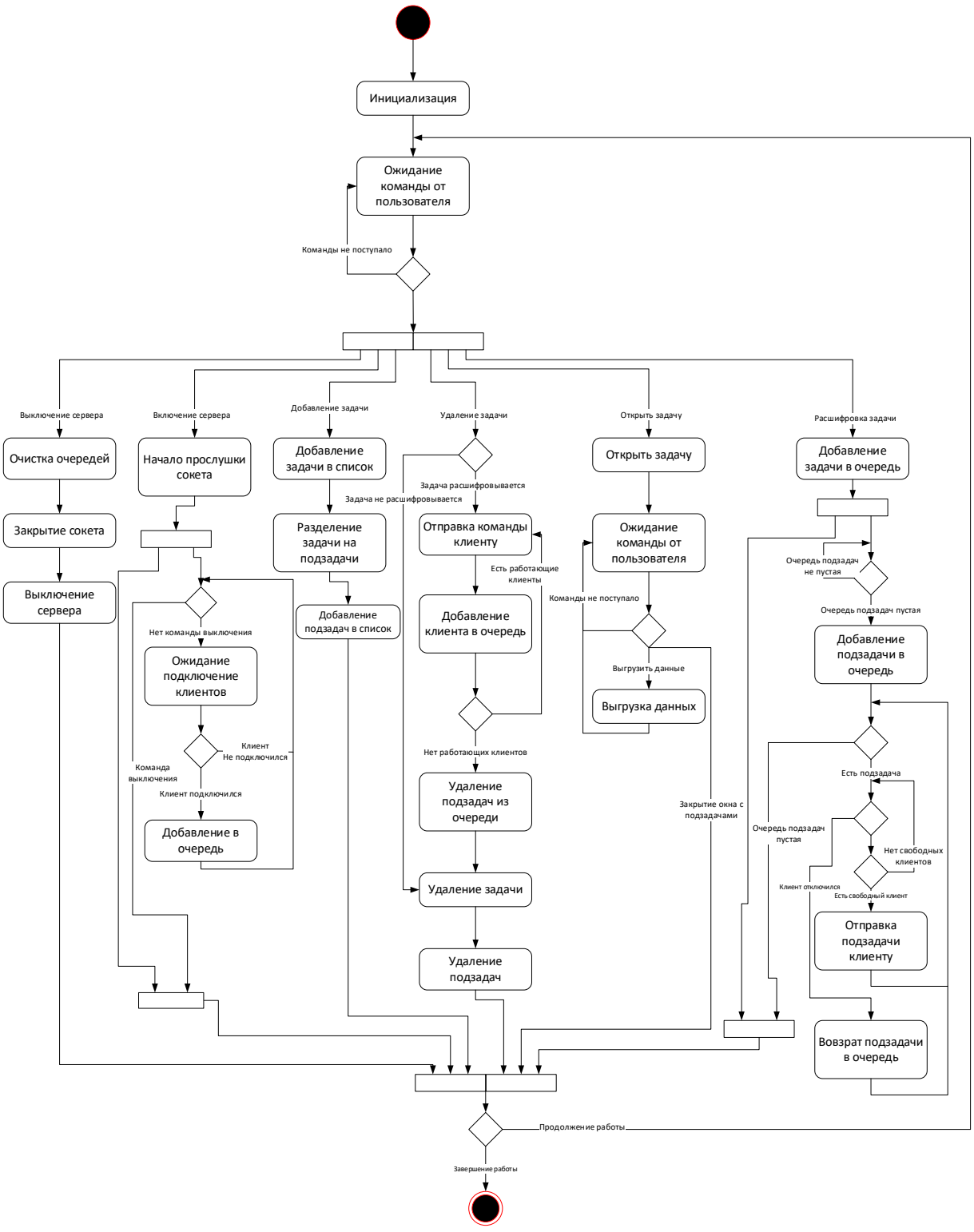


Рисунок 52 - UML-диаграмма деятельности сервера

Приложение Е.6 – UML-диаграмма деятельности клиента

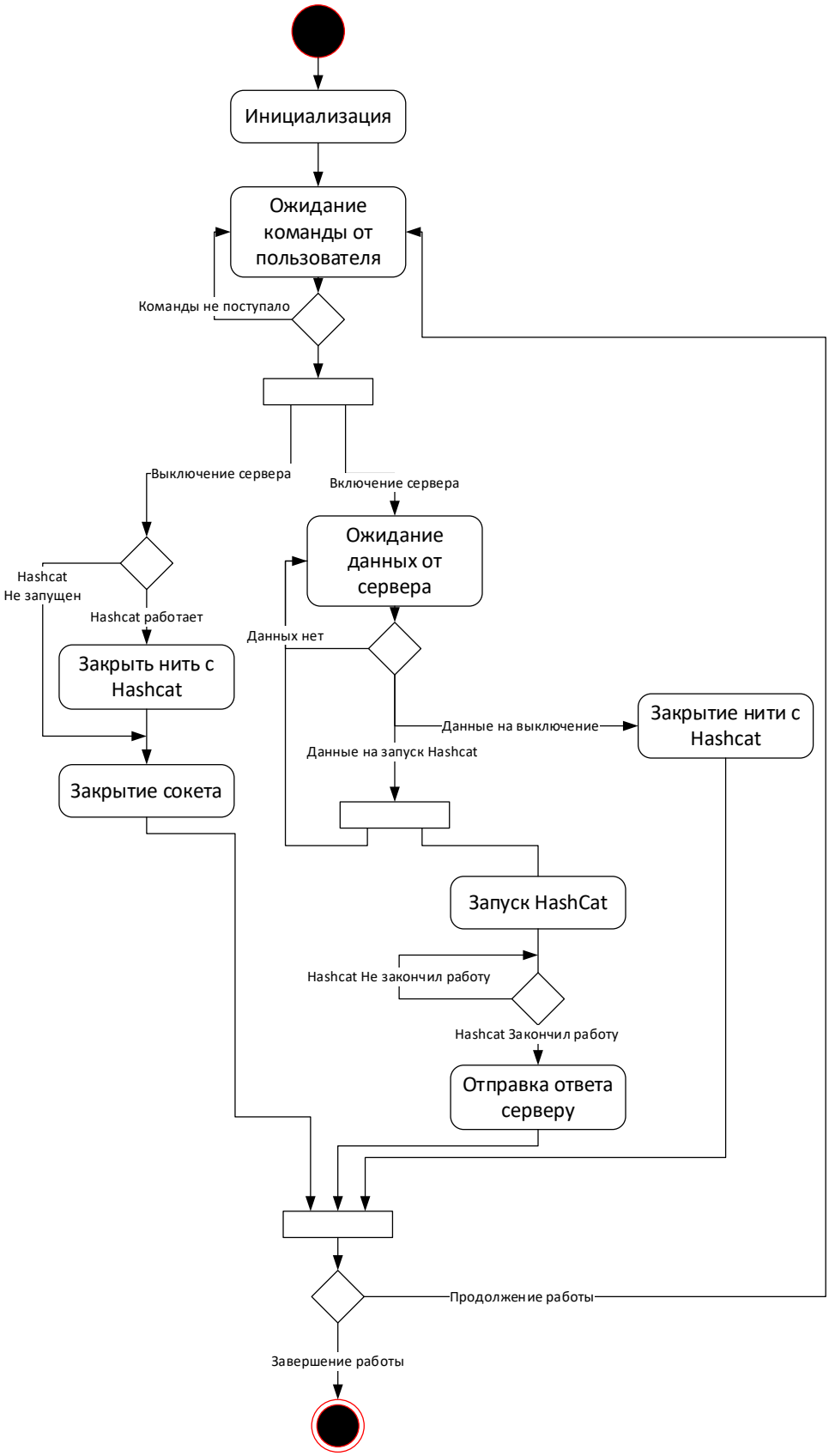


Рисунок 53 - UML-диаграмма деятельности клиента

Приложение Е.7 – UML-диаграмма последовательности

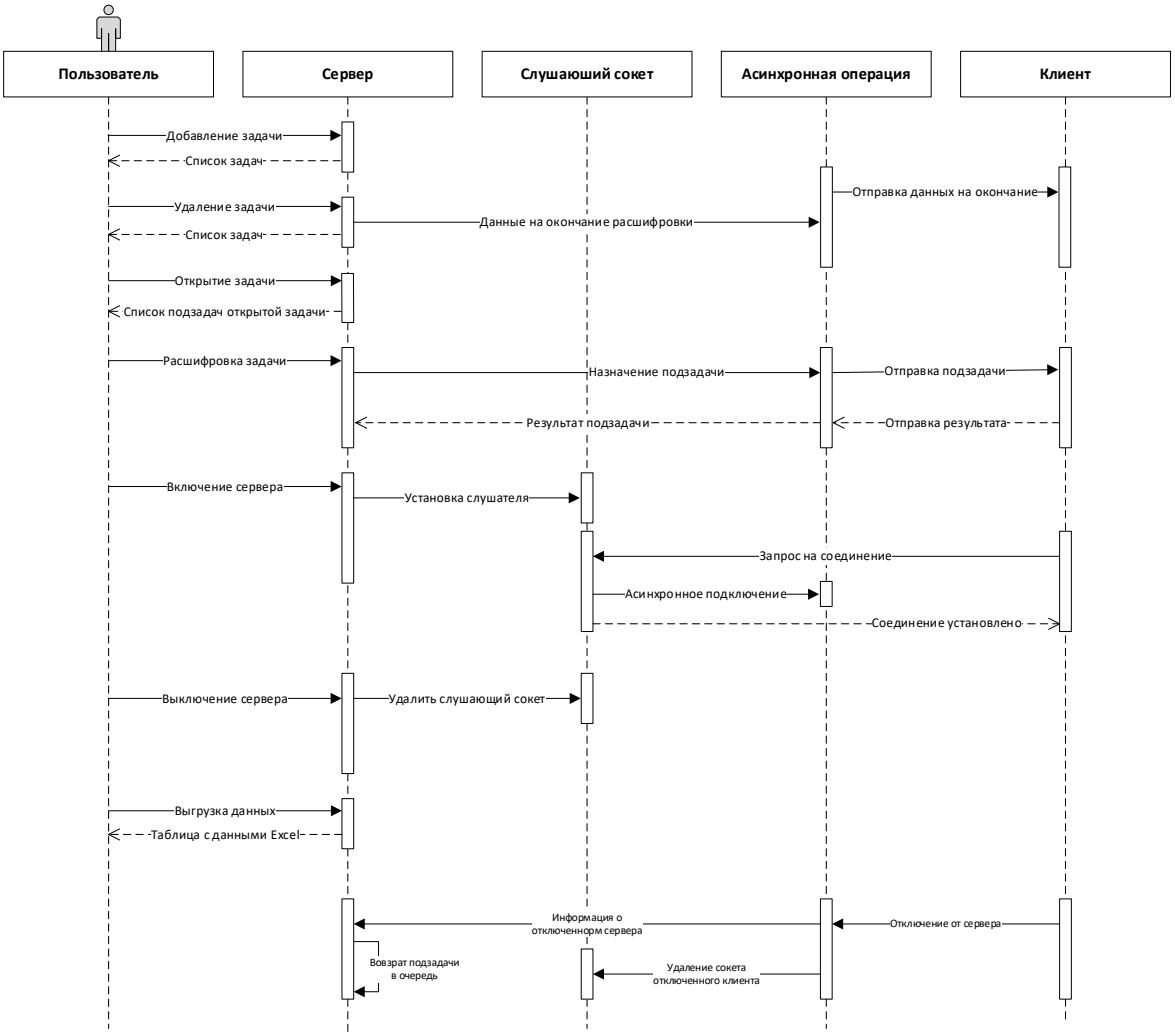


Рисунок 54 - UML-диаграмма последовательности

Приложение Е.8 – UML-диаграмма развертывания

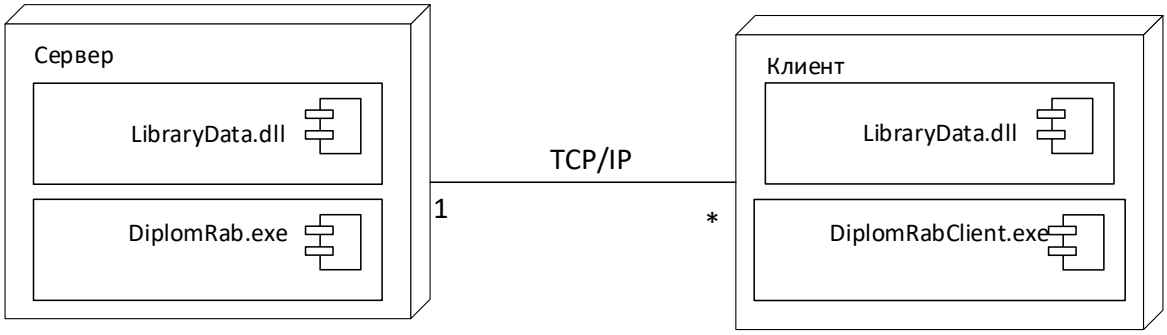


Рисунок 55 - UML-диаграмма развертывания

Приложение F Презентация

ТЕМА РАБОТЫ: «КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ДЛЯ РАСПРЕДЕЛЕННОЙ РАСШИФРОВКИ СВЕРТОК ПАРОЛЕЙ»

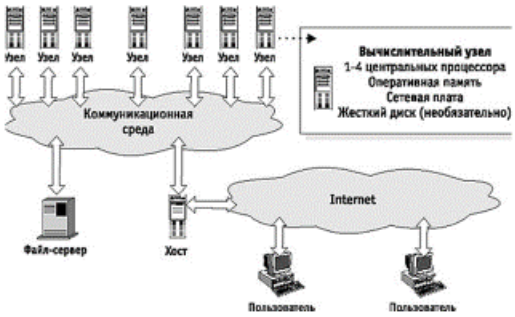
Автор работы: студент группы 16ВВ1 Кривошеев С.В.

Научный руководитель: к.т.н., доцент кафедры ВТ Токарев А.Н.

- Цель работы — рассмотрение существующих систем распределенных вычислений и создание удобной и эффективной в использование распределённой вычислительной клиент-серверной системы кластерного типа
- Объект исследования: клиент-серверный объект распределенных вычислений кластерного типа.
- Предмет исследования: организация процесса восстановления пароля в распределенной вычислительной среде и управления посредством графического интерфейса на серверном приложении.

Распределенная система кластерного типа

- гибкость в выборе языков программирования и средств вычислений,
- ориентированность на существующую инфраструктуру сетей вычислительной техники,
- минимальная стоимость программного и аппаратного обеспечения,
- быстрое и простое развертывание и настройка системы,
- высокая степень приспособляемости к изменяющимся условиям работы,
- высокая надежность системы и невосприимчивость к отказам отдельных узлов и линий связи,



Задача для реализации в распределенной вычислительной системе

- Восстановление забытых паролей пользователя при нерациональном или невозможном изменения его административными методами
- Восстановление зашифрованных данных
- Предоставление разработчикам криптографических алгоритмов информацию об усовершенствовании алгоритма
- Расшифровка зашифрованных вирусами файлов



HASHCAT

Advanced Password Recovery

- Открытый исходный код
- Бесплатная
- Поддержка CPU и GPU
- Занимает не много места на жестком диске
- Быстрая
- Консольное приложение
- Все управление ключами командной строки

Диаграмма последовательности



Диаграмма вариантов использования Клиента



Диаграмма вариантов использования Сервера

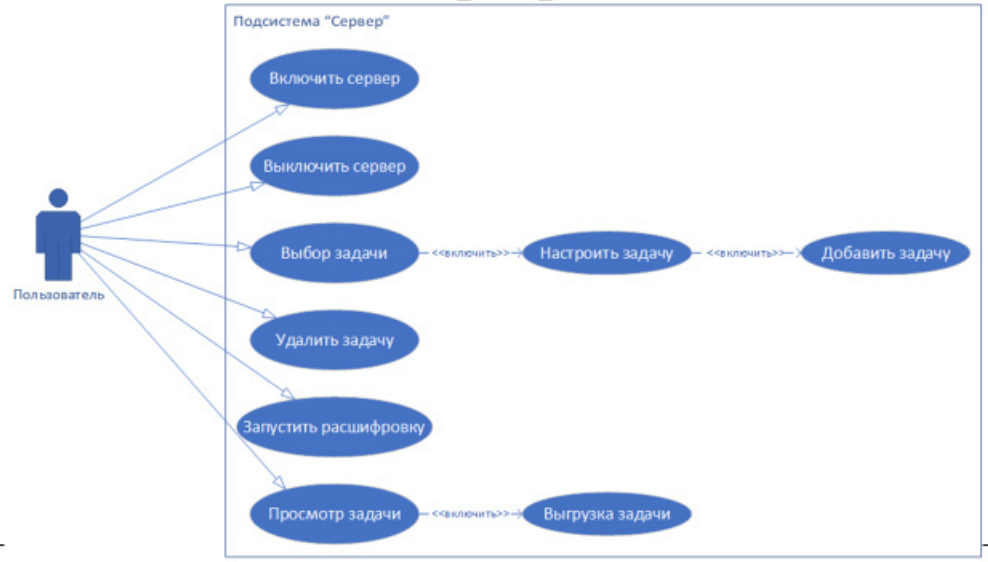


Диаграмма деятельности клиента



Диаграмма деятельности сервера



Диаграмма классов клиента

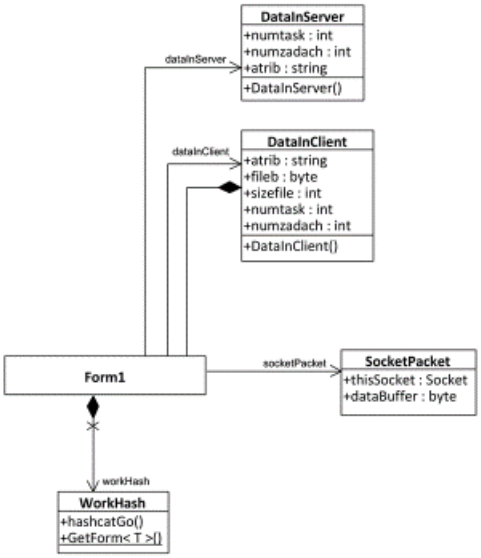
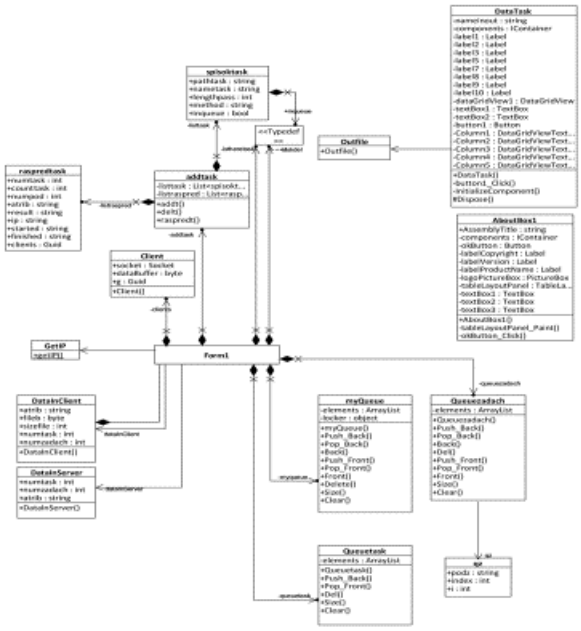
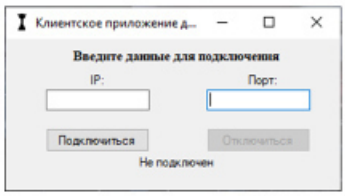
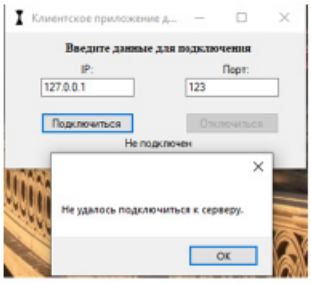


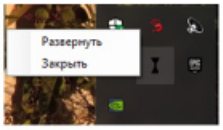
Диаграмма классов сервера

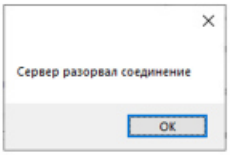


Интерфейс клиентского приложения

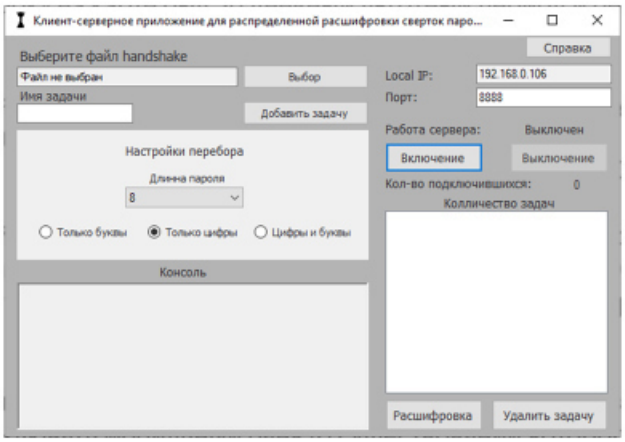




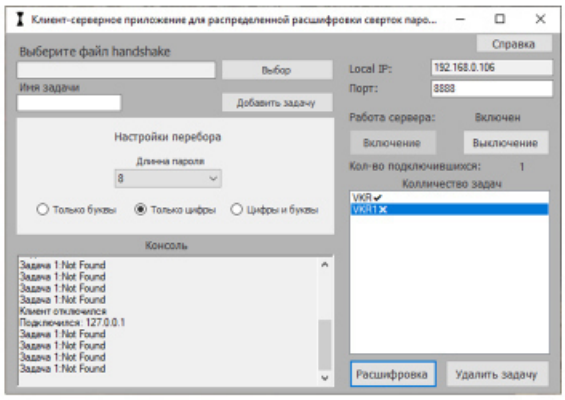


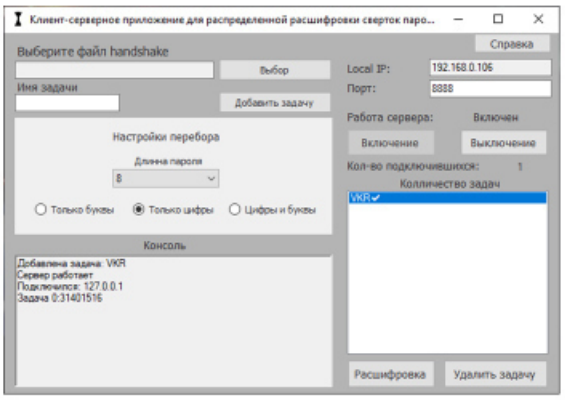


Интерфейс серверного приложения

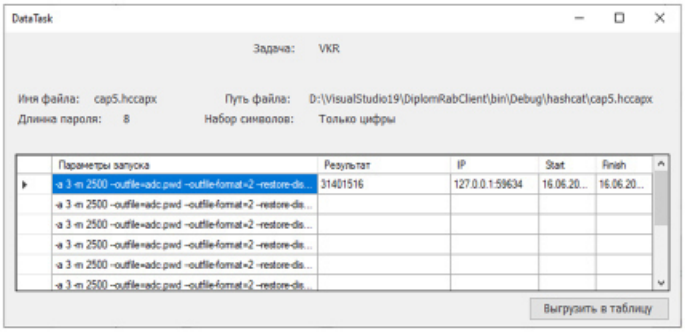


Демонстрация работы программы

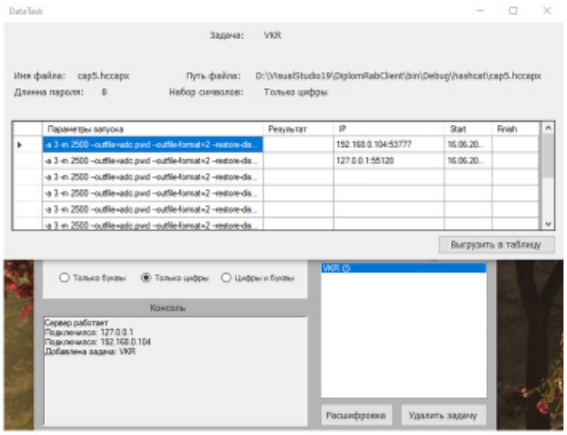


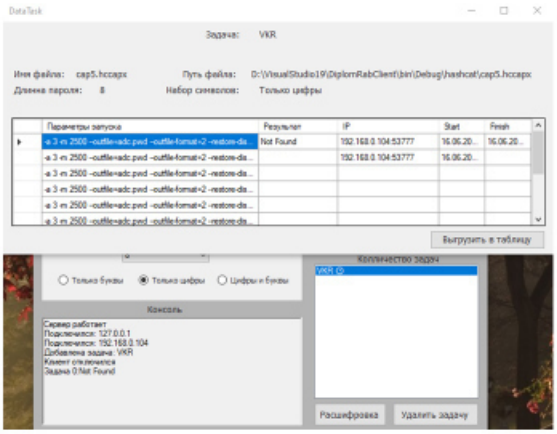


Окно информации о задаче

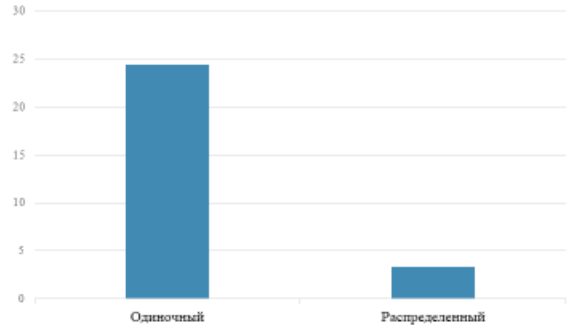


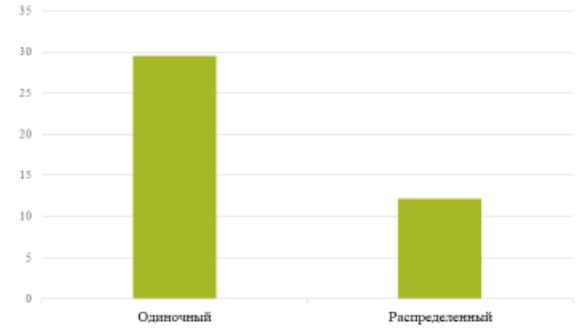
Окно возврата задачи и выдача другому клиенту





Время расчета в разных режимах





№	Режим расшифровки	Время расчета
1	Одиночный	24 минуты 31 секунд
2	Распределенный	3 минуты 30 секунд

№	Режим расшифровки	Время расчета
1	Одиночный	29 минуты 54 секунд
2	Распределенный	12 минуты 16 секунд

Заключение

В результате был создан проект удовлетворяющий поставленным задачам.

- Сервер может распределять подзадачи по клиентам
- При отключении клиента, его подзадача возвращается в очередь
- Реализованы несколько вариантов настройки задачи перед добавлением
- Взаимодействие клиента и сервера может осуществляться по локальной или глобальной сети

СПАСИБО ЗА ВНИМАНИЕ!