## **СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 4

1. АНАЛИЗ ИСХОДНЫХ ДАННЫХ 5

1.1. Анализ предметной области 5

1.2. Специальное и общесистемное программное обеспечение 5

2. ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ 7

2.1. Функции клиента и сервера 7

2.2. Макет пользовательского интерфейса 7

2.3. Прикладной протокол взаимодействия клиента и сервера 9

2.4. Алгоритм работы клиента и севера 9

3. ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 11

3.1. Детальная реализация функциональных частей программного обеспечения 11

3.1.1. Реализация функциональной части клиента и сервера 11

3.1.2. Реализация пользовательского интерфейса клиета 21

3.2. Сопроводительная документация программного обеспечения 24

3.2.1.Описание установки и запуска приложения 24

3.2.2.Руководство пользователя 25

3.3. Тестирование программного обеспечения 25

ЗАКЛЮЧЕНИЕ 29

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 30

## **ВВЕДЕНИЕ**

Данная пояснительная записка содержит в себе описание выполнения курсового проекта, результатом выполнения которого является программа «Удаленный менеджер сервисов». В основе её работы лежит так называемая модель взаимодействия «клиент-сервер», которая позволяет разделить функционал и вычислительную нагрузку между клиентскими приложениями (заказчики услуг) и серверными приложениями (поставщиками услуг).Основная идея данной модели взаимодействия состоит в разделении сетевого приложения на несколько компонентов, каждый из которых реализует специфический набор сервисов. Компоненты такого приложения могут выполняться на разных компьютерах, выполняя серверные и/или клиентские функции. Это позволяет повысить надежность, безопасность и производительность сетевых приложений и сети в целом. Говоря простым языком, разрабатываемая программа функционирует следующим образом: клиент делает запрос серверу, который выполняет его и возвращает результат.

## **1 АНАЛИЗ ИСХОДНЫХ ДАННЫХ**

1.1 Анализ предметной области

Согласно условиям курсового проекта, необходимо разработать клиент-серверное приложение «Удаленный менеджер сервисов». Для начала следует понять, как работают клиент серверные приложения.

**Архитектура клиент-сервер** предъявляет специфические требования, как к приложению-клиенту, так и к приложению-серверу. Программа, удовлетворяющая этим требованиям, может считаться клиент-серверным приложением, выполняющим распределенную обработку данных.

Под распределенной обработкой понимается выполнение серверной частью программы запросов клиентской части. Серверная часть приложения обеспечивает хранение данных и их обработку, а клиентская часть передает серверу соответствующие запросы.

**Преимущества клиент-серверных систем:**

* Клиент-серверный подход – модульный, причем серверные программные компоненты компактны и автономны.
* Поскольку каждый компонент выполняется в отдельном защищенном процессе пользовательского режима, сбой сервера не повлияет на остальные компоненты операционной системы.
* Автономность компонентов делает возможным их выполнение на нескольких процессорах на одном (симметричная многопроцессорная обработка) или на нескольких компьютерах сети (распределенные вычисления).
* Обязанность клиента, как правило – предоставить пользовательские сервисы и, прежде всего, пользовательский интерфейс, то есть средства для приема, отображения и редактирования данных, введенных пользователем, которые служат основой для запроса серверу. Кроме того, клиент можно настроить на обработку части данных, чтобы уменьшить нагрузку на ресурсы сервера.

Помимо того, что разрабатываемая программа должна работать как на одном, так и на нескольких устройствах, сама программа является менеджером сервисов. В зависимости от операционной системы сервисы могут иметь разные названия. Платформой реализации разрабатываемой программы является ОС Windows, следовательно, сервисы в этой ОС принято называть «службами».

Службы – приложения, автоматически (если настроено) запускаемые вместе с системой и выполняющиеся вне зависимости от статуса пользователя.

Также, в условии указано, что приложения клиента и сервера должны соединяться посредством транспортного протокола TCP.

1.2 Специальное и общесистемное программное обеспечение

Разработка программы выполнялась в операционной системе Windows с помощью среды разработки DEV-C++. Выбор языка для разработки обусловлен тем, что C++ поддерживает работу с транспортным протоколом TCP и имеет соответствующие библиотеки для этого, а также поддерживает низкоуровневую работу с памятью.

Клиентский интерфейс представляет собой совокупность оконных форм, но интерфейс приложения сервера является консолью. В качестве вспомогательной библиотеки для создания пользовательского интерфейса использовался Windows API (WinAPI).

Данный выбор обусловлен тем, что WinAPI является одним из лучших вариантов для разработки пользовательского интерфейса на операционной системе Windows по следующим причинам:

Во-первых, Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows.

Во-вторых – работы через данный интерфейс – это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.

## **2 ПРОГРАММНОЕ ПРОЕКТИРОВАНИЕ**

2.1 Функции клиента и сервера

Основные функции приложения клиента:

* Подключение к серверу с последующим обменом информацией с сервером посредством транспортного протокола TCP;
* Предоставление пользователю возможностей просмотра и редактирования полученных данных с помощью клиентского интерфейса;
* Вывод информационных сообщений в стандартный поток вывода на сервере.

Основные функции приложения сервера:

* Предоставление возможности клиентскому приложению получения запрашиваемых данных;
* Обеспечение надежности, согласованности и защищенности предоставляемых данных;
* Выполнение указаний клиента по отношению к получаемым данным.

2.2 Макет пользовательского интерфейса

Программа является оконным приложением, способным работать в соответствии с выбранным режимом. То есть, и приложение клиента, и приложение сервера является одной и той же программой, запущенной в соответствующем режиме. Таким образом, для правильного функционирования необходимо и достаточно, чтобы было запущено не менее двух экземпляров данной программы, один из которых работает как сервер, а второй, соответственно, как клиент.

При работе в любом из режимов окно приложения содержит элементы интуитивно понятные в использовании и позволяющие выполнять следующие функции:

* Переключение между режимами работы.
* Осуществлять ввод необходимых данных для запуска приложения в соответствии с выбранным режимом.
* Осуществлять непосредственно запуск работы приложения.

Макеты главного окна приложения, запущенного в режиме сервера и клиента изображены на рисунках 2.1 и 2.2 соответственно.

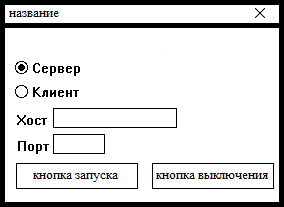


Рисунок 2.1 – интерфейс главного окна (режим сервера)

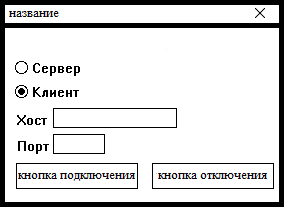


Рисунок 2.2 – интерфейс главного окна (режим клиента)

В случае успешного подключения клиента к серверу, вместо главного окна клиента появляется окно для работы с полученными данными. Данное окно должно содержать такие элементы управления, которые позволят предельно рационально использовать пространство рабочего стола, и при этом предоставлять пользователю возможность просматривать и изменять полученные данные наиболее удобным образом. Макет окна-обозревателя данных представлен на рисунке 2.3.

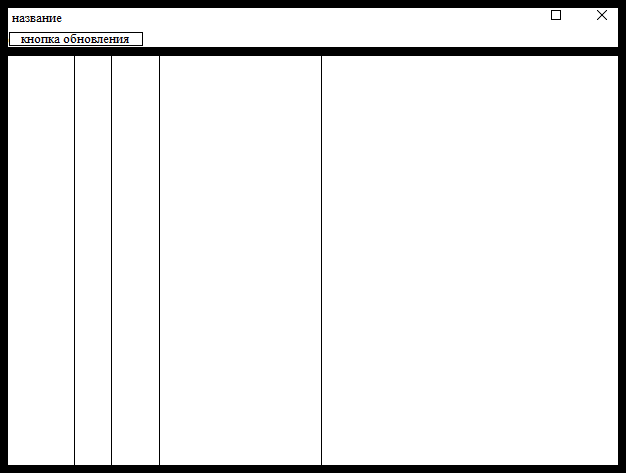


Рисунок 2.3 – макет окна-обозревателя

2.3 Прикладной протокол взаимодействия клиента и сервера

На транспортном уровне стека TCP-IP используется два основных протокола: TCP и UDP. Существуют общие задачи транспортного уровня, с которыми справляются как TCP, так и UDP. Основных задач собственно две: **сегментация данных**, приходящих с уровня приложений и **адресация приложений** (передающего и принимающего) при помощи портов.

Помимо этого, протокол TCP обеспечивает:

* Надежную защиту сегментов;
* Упорядочивание сегментов при получении;
* Работу с сессиями;
* Контроль над скоростью передачи.

Рассмотрим эти возможности более детально.

**Надежная доставка сегментов**

Под надежной доставкой подразумевается автоматическая повторная пересылка недошедших сегментов. Каждый сегмент маркируется при помощи специального поля – порядкового номера (securenumber). После отправки некоторого количества сегментов, TCPна отправляющем узле ожидает подтверждения от получающего, в котором указывается порядковый номер следующего сегмента, который адресат желает получить. В случае, если такое подтверждение не получено, отправка автоматически повторяется. После некоторого количества неудачных попыток, TCP считает, что адресат не доступен, и сессия закрывается.

Таким образом, надежная доставка не означает, что данные дойдут в случае, если кто-то выдернул кабель из коммутатора. Она означает, что разработчик ПО, использующий TCPна транспортном уровне, знает, что если сессия не разорвалась, то всё что он получил отправить будет доставлено получателю без потерь. Существует множество данных, критичных к потере любой порции информации. Например, если вы скачиваете приложение из интернета, то потеря одного байта будет означать, что вы не сможете воспользоваться тем, что скачали. По этой причине многие протоколы уровня приложений используют для транспорта TCP.

**Упорядочивание сегментов при получении**

Как не сложно догадаться, каждый сегмент на нижних уровнях TCP-IPобрабатывается индивидуально. То есть, как минимум, он будет запакован в индивидуальный пакет. Пакеты идут по чети и промежуточные маршрутизаторы в общем случае уже ничего не знают о том, что запаковано в эти пакеты. Часто пакеты с целью балансировки нагрузки могут идти по сети разными путями, через разные пути, через разные промежуточные устройства, с разной скоростью. Таким образом, получатель, декапсулировав их, может получить сегменты не в том порядке, в котором они отправлялись.

TCP автоматически пересоберёт их в нужном порядке используя всё то же поле порядковых номеров и передаст после склейки на уровень приложений.

**Работа с сессиями**

Перед началом передачи полезных данных, TCP позволяет убедиться в том, что получатель существует, слушает нужный отправителю порт и готов принимать данные. Для этого устанавливается сессия при помощи механизма трёхстороннего рукопожатия (three-wayhandshake). Далее, в рамках сессии передаются пользовательские данные. После завершения передачи сессия закрывается, тем самым получатель извещается о том, что данных больше не будет, а отправитель извещается о том, что получатель извещён.

**Контроль над скоростью передачи**

Контроль над скоростью передачи позволяет корректировать скорость отправки данных в зависимости от возможностей получателя. Например, если быстрый сервер отправляет данные медленному телефону, то сервер будет передавать данные с допустимой для телефона скоростью.

Благодаря этому механизму скользящего окна (slidingwindow), TCP может работать с сетями разной надежности. Механизм плавающего окна позволяет менять количество пересылаемых байтов, на которые надо получать подтверждение от адресата. Чем больше размер окна, тем больший объём информации будет передан до получения подтверждения. Для надежных сетей подтверждения можно присылать редко, чтобы не добавлять трафика, поэтому размер окна в таких сетях автоматически увеличивается. Если же TCP видит, что данные теряются, размер окна автоматически уменьшается. Это связано с тем, что если мы передали, например,   
3 килобайта информации и не получили подтверждения, то мы не знаем, какая конкретно часть из них не дошла и вынуждены пересылать все три килобайта заново. Таким образом, для ненадёжных сетей, размер окна должен быть минимальным.

Механизм скользящего окна позволяет TCP постоянно менять размер окна – увеличивать его пока сё нормально и уменьшать, когда сегменты не доходят. Таким образом, в любой момент времени размер окна будет более или менее адекватен состоянию сети.

**Структура TCP**

Заголовок TCP-сегмента имеет следующую структуру:

* Sourceportи Destinationport– это соответственно номера портов получателя и отправителя, идентифицирующие приложения на отправляющем и принимающем узлах.
* Sequencenumberи Acknowledgmentnumber– это порядковый номер сегмента и номер подтверждения, которые используются для надёжной доставки. Например, если отправитель шлёт сегмент с SN 100, то получатель может ответить на него ACK 101 SN 200, что формально означает:«Я получил твой сегмент с номером 100 и жду от тебя 101-го, кстати, у меня своя нумерация. Мои номера начинаются с 200».Отправитель, в свою очередь, может ответить SN101 ACK201, что означает«Я получил от тебя сегмент с номером 200, могу принять следующий 201-ый, а вот тебе мой 101-ый сегмент, которого ты ждешь».Ну и так далее.
* Headerlength – это четырёхбитное поле, содержащее в себе длину заголовка TCP-сегмента;
* Reserved– 6-зарезервированных на всякий случай бит;
* Control– поле с флагами, которые используются в процессе обмена информацией и описывают дополнительное назначение сегмента. Например, флаг FINиспользуется для завершения соединений,а SYNи ACK– для установки;
* Window–содержит размер окна, о котором было сказано выше;
* Checksum – контрольная сумма заголовка и данных;
* Urgent– признак важности (срочности) данного сегмента;
* Options – дополнительное необязательное поле, которое может использоваться, например, для тестирования протокола;
* В разделе данных содержаться собственно данные, полученные от протокола уровня приложений., либо их часть, если данные пришлось разбивать.

…

|  |
| --- |
| Кое-что про протокол поверх TCP/UDP:  #define PROTOCOLID \*((DWORD \*)"\17VS\3")  #define CMD\_ANY 0  #define CMD\_CHECKOUT 0x80000000  #define CMD\_LIST 0x0C000000  #define CMD\_SET 0x18000000  //  // СТРУКТУРА: Datagram  //  // СОДЕРЖИМОЕ: передоваемая комманда и данные  //  struct Datagram {  DWORD id;  DWORD cmd\_cou; // самый старший байт – получаемая/обрабатываемая команда  // младшие 3 – количество служб в списке (в ответе на CMD\_LIST)  char data[];  }; |

**2.4 Алгоритм работы клиента и севера**

//вот тут не смогу точно написать что-либо, не посмотрев на то, что ты сделал

## **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

**3.1 Детальная реализация функциональных частей программного обеспечения**

3.1.1 Реализация функциональной части клиента и сервера

//куча листингов и описаний используемых функций

3.1.2 Реализация пользовательского интерфейса клиета

Согласно разработанным макетам, программный интерфейс состоит из двух окон: главного и обозревателя.

Для реализации всех необходимых функций, главное окно содержит следующие элементы управления:

* Два комбобокса с соответствующими подписями «Сервер» и «Клиент», которые используются для выбора работы программы. По умолчанию выбран режим сервера.
* Два поля с подписями «Хост» и «Шлюз», необходимые для получения от полюзователя данных для запуска сервера либо подключения клиента к нему. Поле ввода хоста доступно только в режиме работы «Клиент» и имеет соответствующий шаблон для ввода.
* Две кнопки, выполняющие функции, соответствующие выбранному режиму работы. При работы программы в режиме «Сервер» они используются для запуска и остановки сервера соответственно. В случае работы в режиме «Клиент» эти же кнопки используются для подключения к серверу. Подписи на кнопках также соответствуют режиму работы программы.

Изображения работы программы в режимах «Сервер» и «Клиент» изображены на рисунках 3.1 и 3.2 соответственно.

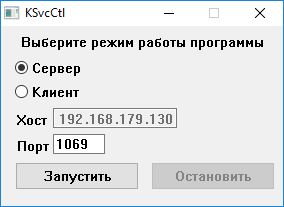


Рисунок 3.1 – главное окно в режиме работы «Сервер»

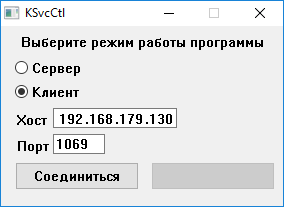


Рисунок 3.2 – главное окно в режиме работы «Клиент»

В целях реализации функционала окна-обозревателя данное окно содержит в себе следующие элементы управления:

* Кнопка «Обновить», которая используется для обновления информации (применения изменений для сервера и отображения информации о службах в соответствующем виде). Также данная кнопка устанавливает размеры остальных элементов управления данного окна в стандартные значения, если они были изменены.
* Так как одной из лучших форм представления информации о системных службах является таблица, окно-обозреватель использует элемент listviewдля её реализации. Таблица содержит колонки: «Внутр. имя» (внутреннее имя службы), «Имя», «Статус» и «Тип запуска».
* Для изменения параметров конкретных служб используется контекстное меню, вызов которого осуществляется посредством нажатия правой кнопки мыши по соответствующей службе. Данное меню позволяет изменить статус службы и её тип запуска, для чего имеет соответствующие пункты.

Внешний вид окна-обозревателя изображен на рисунке 3.3. На рисунке 3.4 изображено контекстное меню в момент его использования.

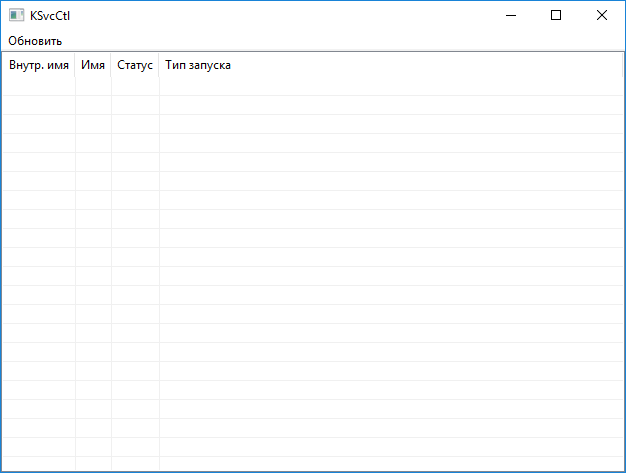


Рисунок 3.3 – окно-обозреватель служб

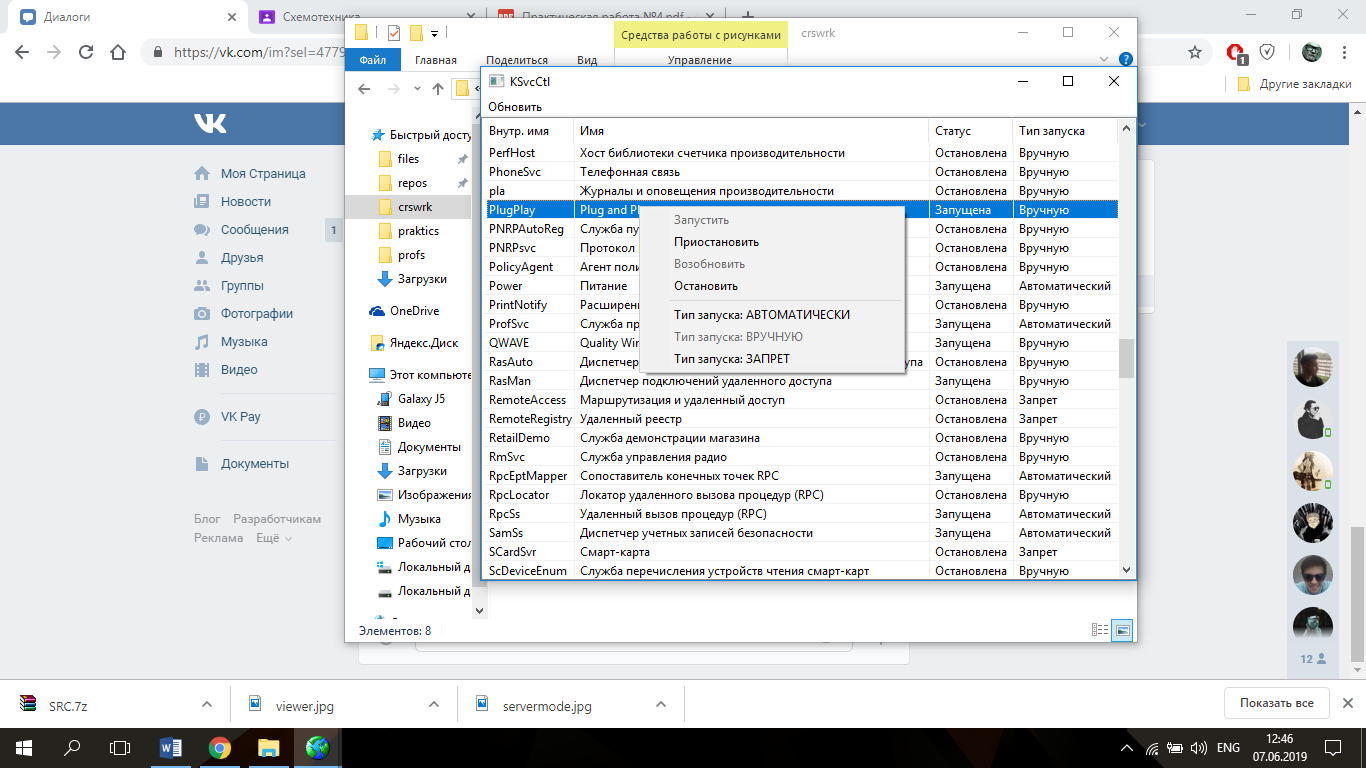


Рисунок 3.4 – возможность управления службами

**3.2.2 Руководство пользователя**

//ну как бы легко тут очень

**3.3 Тестирование программного обеспечения**

//снова черным ящиком?))

**ЗАКЛЮЧЕНИЕ**

В процессе выполнения данного курсового проекта было разработано клиент-серверное приложение «Удаленный менеджер сервисов». В результате выполнения были приобретены знания и умения разработки клиент-серверных приложений, программирования сокетов и сетевых соединений.

**ЛИТЕРАТУРА**

//да