Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**КУРСОВОЙ ПРОЕКТ**

по дисциплине **«Модульное тестирование»**

на тему **«Электронный магазин для компьютерных комплектующих»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Попова Ю.Б.

Минск 2018

Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине **«Модульное тестирование»**

на тему **«Электронный магазин для компьютерных комплектующих»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Попова Ю.Б.

Минск 2018

Введение 4

1. Аналитический обзор состояния вопроса 7

2. Техническое задание на предмет разработки 9

3. Моделирование и проектирование программного обеспечения 10

3.1. Архитектура ПО и описание протокола взаимодействия клиента и сервера 10

3.2. Серверная часть 11

3.3. Клиентская часть 13

4. Реализация программного обеспечения 15

4.1. Руководство для системного администратора 15

4.2. Руководство для пользователя (клиентская часть) 16

5. Развертывание и тестирование программного обеспечения 20

Заключение 21

Список использованных источников 22

ПРИЛОЖЕНИЕ А 23

Графическая часть 23

ПРИЛОЖЕНИЕ Б 28

Исходный код приложения 28

# Введение

Компьютерной (вычислительной) сетью называется совокупность компьютеров (ЭВМ), взаимосвязанных через каналы передачи данных и обеспечивающих пользователей средствами обмена информацией и коллективного использования аппаратных, программных и информационных ресурсов сети.

История развития компьютерных сетей началась в 60-х годах прошлого столетия. Сначала появились многотерминальные системы разделения времени. В таких системах мощная ЭВМ обслуживала одновременно несколько пользователей, имеющих в своем распоряжении терминал (монитор с клавиатурой), с помощью которого он мог вести диалог с ЭВМ. ЭВМ по очереди обрабатывала программы и данные, поступающие с каждого терминала. Терминалы, как правило, рассредоточивались по всему предприятию и функции ввода-вывода информации были распределенными, а ее обработка проводилась только центральной ЭВМ. Подобные многотерминальные централизованные системы внешне напоминали локальные вычислительные сети, до создания которых в действительности нужно было пройти еще большой путь. Затем была решена задача доступа к ЭВМ с терминалов, удаленных от нее на сотни (а то и тысячи) километров. Терминалы в этом случае соединялись с ЭВМ через телефонные линии с помощью модемов. Такие вычислительные сети получили название распределенных или глобальных. Следующим этапом в развитии вычислительных сетей стали соединения не только «терминал — ЭВМ», но и «ЭВМ — ЭВМ». ЭВМ стали обмениваться данными в автоматическом режиме и впервые появились возможности обмена файлами, синхронизации баз данных, использования электронной почты. Иными словами, появились те службы, которые в настоящее время стали традиционными сетевыми сервисами.

Исторически первые компьютерные сети были созданы агентством АRРА по заданию военного ведомства США. В 1969 году министерство обороны США инициировало работы по объединению в единую сеть суперкомпьютеров оборонных и научно-исследовательских центров. Эта сеть, получившая название ARPANET, стала отправной точкой для создания самой известной ныне глобальной сети — Internet.

В 70-х гг. XX века, в связи с развитием микроэлектроники, начался интенсивный процесс распределения вычислительных ресурсов, что впоследствии привело к необходимости обратного объединения всех вычислительных ресурсов в одну систему. Только теперь это объединение происходило уже не на базе одного компьютера, а путем подключения к сети отдельных распределенных компьютеров. Подобные компьютерные сети стали называться локальными компьютерными сетями.

На начальном этапе создания локальных компьютерных сетей для объединения компьютеров использовались самые разнообразные нестандартизированные устройства и программное обеспечение. Создание сети в это время требовало от разработчиков изобретательности и больших усилий. В середине 80-х гг. положение дел в локальных компьютерных сетях стало кардинально меняться в сторону создания стандартных технологий объединения компьютеров в единую сеть. Были разработаны специальные методы и правила обмена информацией между компьютерами, среди которых наиболее известными стали стандарты Ethernet, Toking Ring, FDDI, Arcnet. В этих стандартах были строго регламентированы длина, вид и порядок следования кодов, посылаемых компьютерами в сеть, правила доступа к сети отдельных компьютеров и т.д. Кроме того, в это время начали интенсивно использоваться стандартные персональные компьютеры, которые очень быстро потеснили мини-ЭВМ и мэйнфреймы. Разработанные стандартные сетевые технологии, а также использование персональных компьютеров значительно упростили процесс создания компьютерных сетей. Для создания сети достаточно стало приобрести специальные сетевые платы (сетевые адаптеры) соответствующего стандарта, стандартный кабель со стандартными разъемами и установить на компьютер сетевую операционную систему.

На сегодняшний день основными направлениями использования компьютерных сетей являются следующие:

1. Совместный доступ к аппаратным, программным и информационным ресурсам (использование дисков или только определенных папок и файлов других компьютеров, принтеров, программного обеспечения, баз и банков данных);
2. Предоставление коммуникационных услуг (службы информации, электронная почта, телеконференции и т.д.);
3. Распределенная обработка данных (сети можно использовать для обработки данных на отдельных компьютерах, связанных между собой и представляющих распределенную систему).

В общем случае любая компьютерная сеть состоит из набора трех основных компонентов:

* рабочих станций (персональных компьютеров пользователей);
* файлового сервера (главного компьютера сети);
* сети или средств передачи данных (физической передающей среды и АПД), обеспечивающих обмен информацией между компьютерами.

В частном случае компьютерная сеть может содержать несколько серверов разной степени иерархии или состоять только из двух или более однотипных рабочих станций.

Функциональные возможности сети определяются услугами, которые она предоставляет. Для реализации каждой из услуг сети и доступа пользователя к этой услуге используется сетевое программное обеспечение.

В настоящее время распространены две основные концепции построения сетевого ПО.

Первая концепция ориентирована на предоставление многим пользователям ресурсов главного компьютера сети — файлового сервера. Управление ресурсами файлового сервера и предоставление к ним доступа производится сетевой операционной системой. Ее основная часть находится на файловом сервере, а на рабочих станциях (компьютерах пользователей) устанавливается только небольшая оболочка, выполняющая роль интерфейса между программами, обращающимися за ресурсами к файловому серверу. Рабочие станции используют программы или данные файлового сервера, а также другие его ресурсы (принтер, модем и т.п.). Программы файлового сервера могут использоваться всеми пользователями одновременно, но для выполнения модули этих программ по мере необходимости переносятся на рабочую станцию. При этом вся обработка данных, даже если они являются общими ресурсами и хранятся на файловом сервере, происходит непосредственно на рабочих станциях (очевидно, что для этого файлы, в которых хранятся данные, должны быть перемещены на рабочую станцию).

Во второй концепции, называемой архитектурой «клиент-сервер», ПО не только обеспечивает коллективное использование ресурсов, но и ориентировано на их обработку в местах размещения ресурсов по запросам пользователей. Программные системы архитектуры клиент-сервер состоят из двух частей: программного обеспечения сервера и программного обеспечения пользователя-клиента. Работа организуется следующим образом: программы-клиенты выполняются на компьютере пользователя и посылают запросы к программе-серверу, которая работает на компьютере общего доступа. Основная обработка данных производится мощным сервером, а на компьютер пользователя посылаются только результаты выполнения запроса.

В приложениях глобальных сетей архитектура клиент-сервер является основной. Широко известны Web-серверы, обеспечивающие хранение и обработку гипертекстовых страниц, FTP-серверы, серверы электронной почты и множество других. Клиентские программы перечисленных служб позволяют сформулировать запрос на получение услуги со стороны этих серверов и принять от них ответ.

# Требования к программному продукту

Миллионы людей каждый день выходят в сеть Интернет, чтобы почитать новости, пообщаться с друзьями, получить полезную информацию, совершить покупку или оплатить счет. Но большая часть рядовых пользователей даже не догадывается о том, как и с помощью чего они всё это делают, да на самом деле большинству людей это и не нужно, главное, чтобы они получали услугу вовремя и качественно.

Здесь мы разберемся с концепцией, которая позволяет нам выполнять все эти действия в сети Интернет. Данная концепция получила название «**клиент-сервер**». Как понятно из названия, в данной концепции участвуют две стороны: [клиент](http://zametkinapolyah.ru/servera-i-protokoly/chto-takoe-klient-klientskij-kompyuter-i-klientskoe-prilozhenie.html) и [сервер](http://zametkinapolyah.ru/servera-i-protokoly/chto-takoe-server.html). Здесь всё как в жизни: **клиент – это заказчик той или иной услуги**, а **сервер – поставщик услуг**. Клиент и сервер физически представляют собой программы, например, типичным клиентом является браузер. В качестве сервера можно привести следующие примеры: все [HTTP сервера](http://zametkinapolyah.ru/servera-i-protokoly/http-server-ili-veb-server.html) (в частности [Apache](http://zametkinapolyah.ru/servera-i-protokoly/server-apache-ustanovka-na-windows.html)), [MySQL сервер](http://zametkinapolyah.ru/zametki-o-mysql/server-mysql/mysql-server-fajl-konfiguracij-my-ini-nastrojka-kodirovki-mysql-servera-tablicy-perekodirovok.html), локальный [веб-сервер AMPPS](http://zametkinapolyah.ru/zametki-o-poleznyx-programmax/lokalnyj-veb-server-ampss.html) или готовая [сборка Denwer](http://zametkinapolyah.ru/zametki-o-poleznyx-programmax/ustanovka-denvera-nastrojka-denvera-sozdanie-sajta-na-denvere.html) (последних два примера – это не проста сервера, а целый набор серверов).

**Клиент и сервер** взаимодействую друг с другом в сети Интернет или в любой другой компьютерной сети при помощи различных сетевых протоколов, например, IP протокол, [HTTP протокол](http://zametkinapolyah.ru/servera-i-protokoly/vsyo-chto-pro-http-protokol.html), FTP и другие. Протоколов на самом деле очень много и каждый протокол позволяет оказывать ту или иную услугу. Например, при помощи HTTP протокола браузер отправляет специальное [HTTP сообщение](http://zametkinapolyah.ru/servera-i-protokoly/http-soobshheniya-struktura-i-sintaksis.html), в котором указано какую информацию и в каком виде он хочет получить от сервера, сервер, получив такое сообщение, отсылает браузеру в ответ похожее по структуре сообщение (или несколько сообщений), в котором содержится нужная информация, обычно это [HTML документ](http://zametkinapolyah.ru/verstka-sajtov/struktura-html.html).

Также стоит заметить, что **в основе взаимодействия клиент-сервер лежит принцип того, что такое взаимодействие начинает клиент**, сервер лишь отвечает клиенту и сообщает о том может ли он предоставить услугу клиенту и если может, то на каких условиях. Клиентское программное обеспечение и серверное программное обеспечение обычно установлено на разных машинах, но также они могут работать и на одном компьютере.

Многие сетевые протоколы построены на архитектуре клиент-сервер, поэтому в их основе обычно лежат одинаковые или схожие принципы взаимодействия, а разницу мы видим лишь в деталях, которые обусловлены особенностями и спецификой области, для которой разрабатывался тот или иной сетевой протокол.

Преимуществом модели взаимодействия клиент-сервер является то, что программный код клиентского приложения и серверного разделен. Если мы говорим про локальные компьютерные сети, то к преимуществам архитектуры клиент-сервер можно отнести пониженные требования к машинам клиентов, так как большая часть вычислительных операций будет производиться на сервере, а также архитектура клиент-сервер довольно гибкая и позволяет администратору сделать локальную сеть более защищенной.

К недостаткам модели взаимодействия клиент-сервер можно отнести то, что стоимость серверного оборудования значительно выше клиентского. Сервер должен обслуживать специально обученный и подготовленный человек. Если в локальной сети ложится сервер, то и клиенты не смогут работать.

В качестве заключения нам стоит явно акцентировать внимание на том, что **архитектура клиент-сервер не делит машины на только клиент или только сервер**, а скорее позволяет распределить нагрузку и разделить функционал между клиентской частью и серверной.

# Проектирование программных модулей

Требуется разработать клиент-серверное приложение на языке C# с графическим интерфейсом для ОС MS Windows, использующее сокеты для сетевого взаимодействия.

Необходимо реализовать поиск файла, считывание атрибутов файла, изменение атрибутов файла, подсчет числа файлов в каталоге и получение размера каталога.

Клиент и сервер на транспортном уровне должны использовать протокол TCP. Серверной части приложения необходимо различать запросы от клиента, а клиенту необходимо «уметь» формировать запрос, поэтому нужно разработать собственный протокол взаимодействия клиента и сервера. Для этого можно создать перечисление, каждый элемент которого будет означать тот или иной запрос.

Для того, чтобы сервер мог работать с запросами от разных клиентов одновременно, он должен быть асинхронным либо многопоточным. В данном случае удобнее будет использовать пул потоков для взаимодействия с клиентами, размер которого задает системный администратор при запуске сервера.

Серверная часть должна работать с файлами, поэтому она будет активно взаимодействовать с файловой системой. Для работы с файловой системой на языке С# нужно использовать пространство имен System.IO.

Так как с сервером работает системный администратор, то серверную часть приложения можно сделать консольной.

Клиентская часть должна быть с удобным, интуитивно понятным и дружелюбным к пользователю графическим интерфейсом. Приложение может использовать Windows Form для графического интерфейса. Основная задача реализации клиентской части – обеспечение требуемого от приложения функционала взаимодействия с сервером и представление информации в удобном для пользователя виде.

# Реализация программных модулей

## Архитектура ПО и описание протокола взаимодействия клиента и сервера

Данное приложение использует архитектуру «клиент-сервер». На транспортном уровне используется протокол TCP. Для сетевого взаимодействия используются поточные сокеты (класс Socket из пространства имен System.Net.Sockets).

Для приема и отправки информации, а также для ее преобразования используется пользовательский класс Courier, который агрегирует сокет. Класс Courier включает в себя методы приема и отправки целых чисел и строк. Подробная UML-диаграмма данного класса представлена на рис. 3.1.1.



Рис. 3.1.1. – UML-диаграмма класса Courier

Для определения запросов сервера используется пользовательское перечисление ClientRequests, в котором каждый элемент означает конкретный запрос клиента серверу:

1. OpenFolderBrowser – дать список доступных устройств.
2. ShowSubFolders – дать список подпапок для данного пути.
3. SelectFolder – выбрать папку и отправить список всех доступных файлов, которые находятся в данной папке.
4. GetAttributes – дать атрибуты выбранного файла.
5. SetAttributes – установить новые атрибуты для данного файла.
6. CloseConnection – завершить соединение

На рис. 3.1.2 приведена UML-диаграмма данного перечисления.



Рис. 3.1.2. – UML-диаграмма перечисления ClientRequests

Также в приложении используется утилитный класс PathUtility, который облегчает работу с путями к файлам и папкам. Его UML-диаграмма представлена на рис. 3.1.3.



Рис. 3.1.3. – UML-диаграмма класса PathUtility

## Серверная часть

Сервер использует список потоков для взаимодействия с множеством клиентов одновременно. Максимально допустимое количество одновременно обслуживаемых клиентов задается системным администратором при запуске программы. Для каждого подключившегося клиента сервер создает поток и добавляет этот поток в список.

На сервере, помимо основного потока и списка потоков, работает еще один поток, который наблюдает за состоянием потоков в списке. Если поток мертв – наблюдающий поток удаляет его из списка потоков и, таким образом, освобождается место для подключения новых клиентов. Наблюдающий поток использует метод MonitorThreads() пользовательского класса ThreadMonitor, UML-диаграмма которого представлена на рис. 3.2.1.



Рис. 3.2.1. – UML-диаграмма класса ThreadMonitor

Основная логика взаимодействия сервера с отдельным клиентом содержится в методе RunService() пользовательского класса ClientObject. Этот метод используется в потоках, созданных для клиента. На рис. 3.2.2 изображена UML-диаграмма класса ClientObject. Данный класс содержит все необходимые методы для взаимодействия с клиентом.



Рис 3.2.2. – UML-диаграмма класса ClientObject

За работу с устройствами, папками и файлами отвечает пользовательский класс FileManager. С помощью данного класса можно получить список файлов, папок и доступных устройств. Его UML-диаграмма изображена на рис. 3.2.3. Также данный класс имеет методы для нахождения размера папки и представлении его в удобном (строковом) виде.



Рис. 3.2.3. – UML-диаграмма класса FileManager

## Клиентская часть

Клиентская часть использует графический интерфейс для взаимодействия с пользователями. На главной форме (рис. 3.3.1) находится listBox, в который выводится список файлов, поисковая строка и другие компоненты.

Рис. 3.3.1. – Схематичное изображение главной формы

Для просмотра и выбора папки используется еще одна форма (рис. 3.3.2).



Рис 3.3.2. – Схематичное изображение формы для выбора папки

Изображение формы для просмотра и изменения атрибутов выбранного файла приведено на рис. 3.3.3.



Рис 3.3.3. – Схематичное изображение формы для изменения атрибутов файла

При нажатии на какую-либо из кнопок, при котором должно происходить сетевое взаимодействие, выполняется сначала проверка соединения. Проверка соединения выполняется так: пытаемся серверу отправить массив байтов, состоящий из 0 байт, если удалось отправить, значит с соединением все в порядке. Если произошла ошибка, то программа уведомит об этом пользователя.

# Проектирование модульных тестов

## Построение ориентированных графов Мак-кейба

Сервер можно запустить через командную строку (или с помощью Windows PowerShell) без параметров (рис. 4.1.1) и с одним параметром (рис. 4.1.2), который означает максимальное количество одновременно обслуживаемых клиентов.

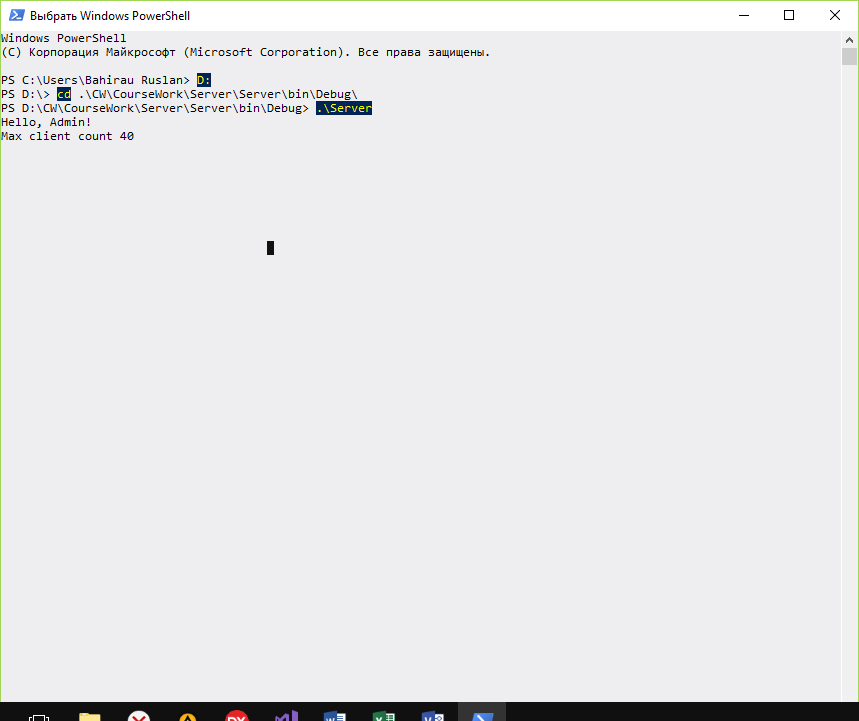


Рис. 4.1.1 – Запуск сервера без параметров

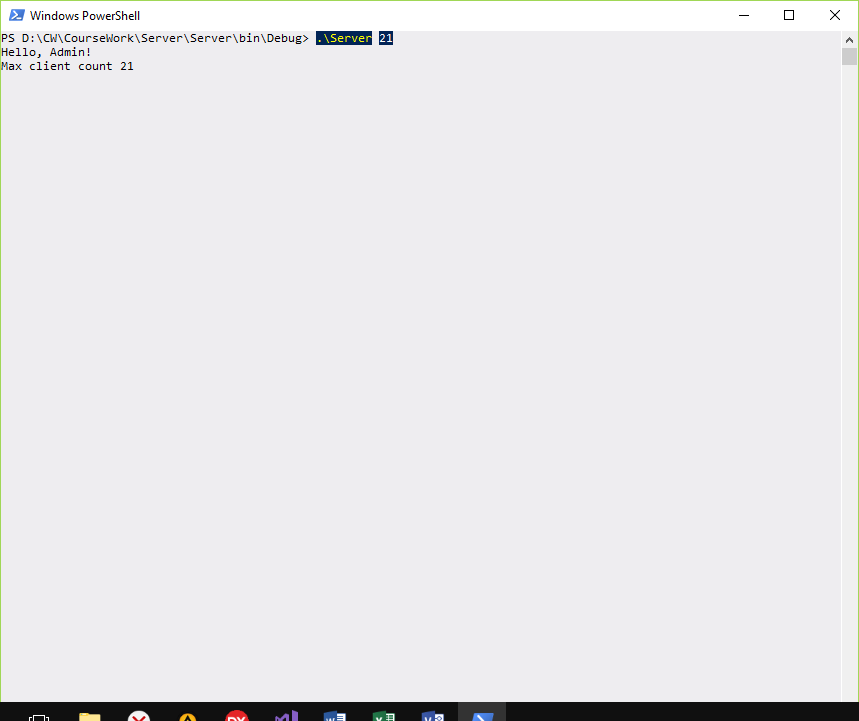


Рис. 4.1.2. – Запуск сервера с параметром, указывающим максимальное количество одновременно обслуживаемых клиентов

Теперь в командной строке будет отображаться информация о новых клиентах, о прекращении соединения и об ошибках (рис. 4.1.3).

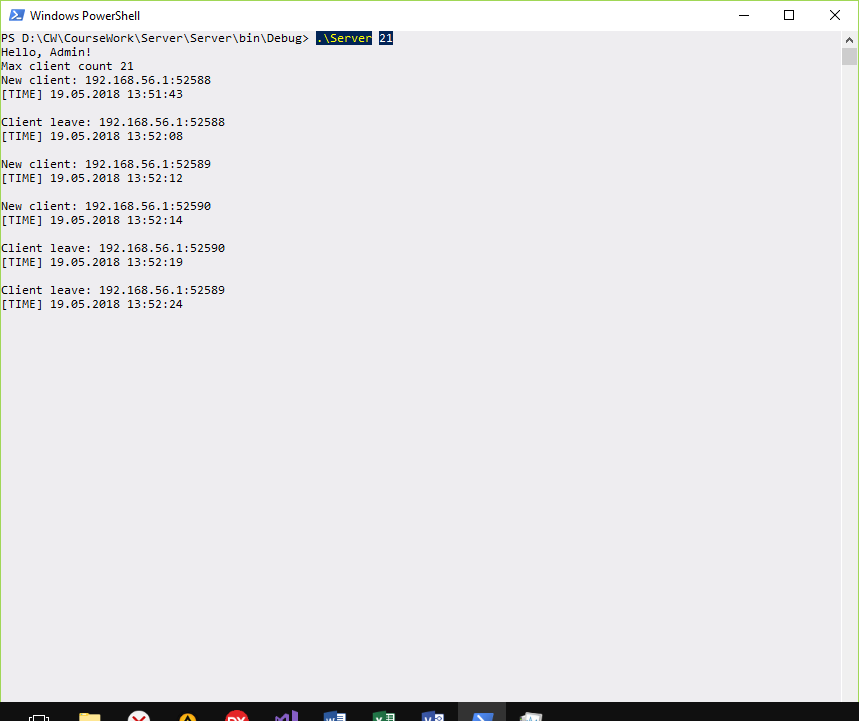


Рис. 4.1.3 – Работа сервера с клиентами

## Формирование проходов

Запустить программу можно просто выполнив файл Client.exe. Появится главное окно приложения (рис. 4.2.1).

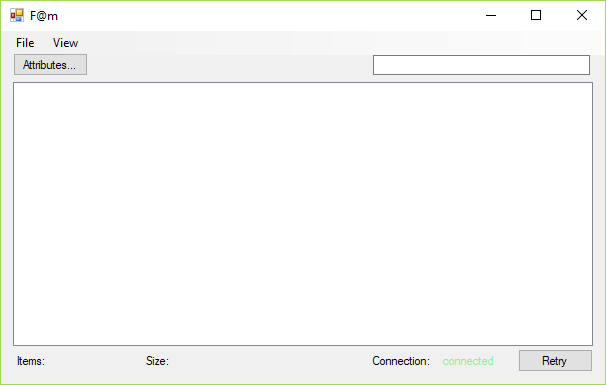


Рис. 4.2.1. – Главное окно приложения

Снизу есть три показателя: «Items», «Size» и «Connection». «Items» показывает количество файлов, доступных пользователю на главной форме. «Size» показывает размер текущей папки. «Connection» показывает состояние соединения с сервером.

Выбрав «File -> Open» откроется окно для выбора папки (рис. 4.2.2). Щелкните два раза по папке, к которой хотите перейти. Нажмите «Select Folder» чтобы выбрать папку. После этих действий на главной форме появится список файлов (рис. 4.2.3).

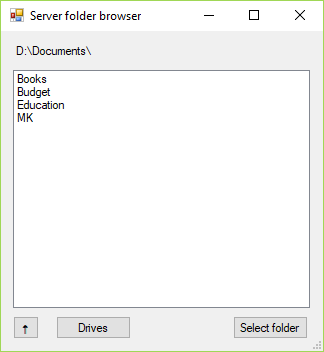


Рис. 4.2.2. – Выбор папки

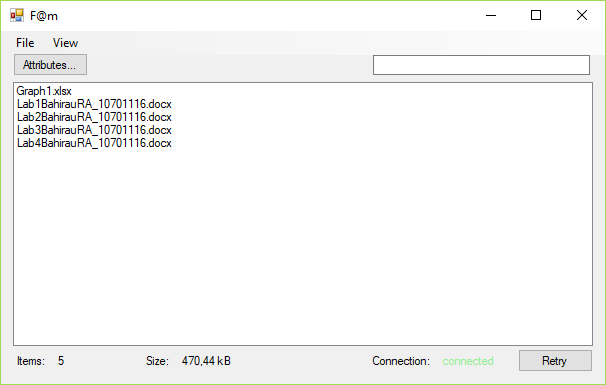


Рис. 4.2.3. – Список файлов на форме

Теперь, если вы хотите найти имя файла с определенной подстрокой, просто введите ее в поле для поиска (рис. 4.2.4).

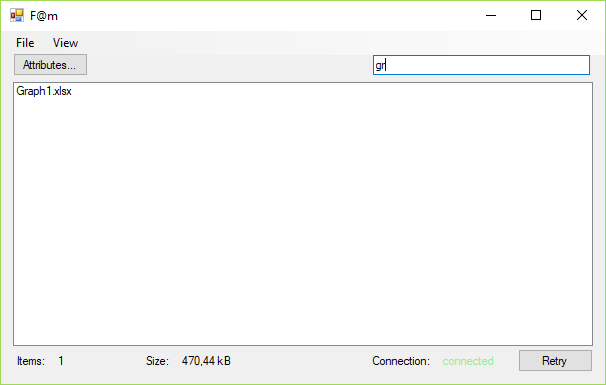


Рис. 4.2.4. – Поиск файлов из списка с подстрокой «gr»

Выберите файл из списка, щелкнув по его имени, и нажмите кнопку «Attributes…». Откроется окно для просмотра и изменения атрибутов файла (рис. 4.2.5).

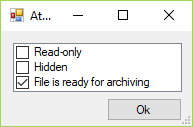


Рис. 4.2.5. – Окно атрибутов файла

Просто выберите и измените нужные атрибуты и нажмите на кнопку «Ok». Готово, теперь атрибуты изменены.

Если нет соединения с сервером или оно потеряно, то появится сообщение (рис 4.2.6).

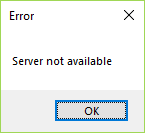


Рис. 4.2.6. – Сообщение о потере соединения

Для попытки подключения к серверу просто нажмите на кнопку «Retry». Если соединение возобновилось, то индикатор «Connection» на главной форме покажет текст салатового цвета.

# Развертывание и тестирование программного обеспечения

# Заключение

При выполнении данного курсового проекта мною были освоены приемы работы с поточными сокетами и разработан простой протокол взаимодействия между клиентом и сервером. Я ознакомился с классами пространств имен System.Net, System.Net.Sockets, System.IO, System.Threading и т.д. На практике закрепил умение разрабатывать многопоточные приложения и приложения, использующие графический интерфейс. Также я узнал больше об атрибутах файла/папки и как с ними работать.

Результатом выполнения данного курсового проекта стала реализация на практике умений разрабатывать многопоточные клиент-серверные приложения, использующие поточные сокеты для сетевого взаимодействия.

# Список использованных источников

1. Олифер, В. Компьютерные сети. Принципы, технологии, протоколы: учебник для вузов./Олифер В., Олифер Н. – 5-е изд. – СПб.: Питер, 2016. – 992с.
2. Пространство имен System.IO. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.io(v=vs.110).aspx.
3. System.Net – пространства имен. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/gg145039(v=vs.110).aspx.
4. Пространство имен System.Net.Sockets. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.net.sockets(v=vs.110).aspx.

# ПРИЛОЖЕНИЕ А

## Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | |
|  |  |  | |  | |  | | КП—10701116 03–ДО–201 | | | | | | | | |
|  |  |  | |  | |  | |
| Изм. | Лист | № документа | | Подпись | | Дата | |
| Разраб. | | Багиров Р.А. | |  | |  | | Архитектура приложения | | Лит | | | | Лист | | Листов |
| Руковод. | | Белова С.В. | |  | |  | |  | Д | |  | **1** | | **5** |
| Консульт. | | Белова С.В. | |  | |  | | 1 40 01 01 БНТУ  г.Минск | | | | | | |
| Н.контр. | | Белова С.В. | |  | |  | |
| Зав.каф. | |  | |  | |  | |
|  | | | | | | | | | | | | | | | | | |
|  | | | Фамилия | | Подпись | | Дата | | *Диаграмма классов серверной части приложения* | | | Лист | | | Листов | | |
| Студент | | | Багиров Р.А. | |  | |  | |
| Руководитель | | | Белова С.В. | |  | |  | | *2* | | | *5* | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Диаграмма классов клиентской части приложения* | Лист | Листов |
| Студент | Багиров Р.А. |  |  |
| Руководитель | Белова С.В. |  |  | *3* | *5* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Интерфейс пользователя* | Лист | Листов |
| Студент | Багиров Р.А. |  |  |
| Руководитель | Белова С.В. |  |  | *4* | *5* |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Блок-схема алгоритма взаимодействия сервера с клиентом* | Лист | Листов |
| Студент | Багиров Р.А. |  |  |
| Руководитель | Белова С.В. |  |  | *5* | *5* |

# ПРИЛОЖЕНИЕ Б

## Исходный код приложения

MainServer.cs

using System;

using System.Net;

using System.Net.Sockets;

using System.Threading;

using System.Collections.Generic;

using MyAddedTypes;

namespace Server

{

class MainServer

{

private static int PORT = 5555;

private static int MAX\_CLIENT\_COUNT = 40;

static void Main(string[] args)

{

try

{

int clientCount = getClientCount(args);

Console.WriteLine("Hello, Admin!\nMax client count {0}",

clientCount);

Socket listener = new Socket(AddressFamily.InterNetwork,

SocketType.Stream, ProtocolType.Tcp);

listener.Bind(new IPEndPoint(IPAddress.Any, PORT));

listener.Listen(clientCount);

Run(listener, clientCount);

listener.Shutdown(SocketShutdown.Both);

listener.Close();

}

finally

{

Console.Write(@"Press 'Enter' to exit: ");

Console.ReadLine();

}

}

private static void Run(Socket listener, int clientCount)

{

List<Thread> threads = new List<Thread>();

Thread monitor = new Thread(new ThreadStart(new ThreadMonitor(threads).MonitorThreads));

monitor.Start();

while (true)

{

if (threads.Count < clientCount)

{

Socket clientSocket = listener.Accept();

Console.WriteLine("New client: {0}", clientSocket.RemoteEndPoint);

Console.WriteLine("[TIME] {0}\n", DateTime.Now);

monitor.Suspend();

threads.Add(new Thread(new ThreadStart(

new ClientObject(clientSocket).RunService)));

threads[threads.Count - 1].Start();

monitor.Resume();

}

}

}

private static int getClientCount(string[] args)

{

int clientCount = MAX\_CLIENT\_COUNT;

if (args.Length > 0)

{

int.TryParse(args[0], out clientCount);

}

return clientCount;

}

}

}

ClientObject.cs

using System;

using System.IO;

using System.Net.Sockets;

using MyAddedTypes;

namespace Server

{

class ClientObject

{

private Courier courier;

public ClientObject(Socket socket)

{

courier = new Courier(socket);

}

public void RunService()

{

while (true)

{

try

{

ClientRequests request = WaitRequest();

switch (request)

{

case ClientRequests.OpenFolderBrowser:

SendOpenFolderBrowser();

break;

case ClientRequests.ShowSubFolders:

SendSubFolders(courier.ReceiveString(),

bool.Parse(courier.ReceiveString()));

break;

case ClientRequests.SelectFolder:

SendFiles(courier.ReceiveString(),

bool.Parse(courier.ReceiveString()));

break;

case ClientRequests.GetAttributes:

SendAttributes(courier.ReceiveString());

break;

case ClientRequests.SetAttributes:

SetAttributes(courier.ReceiveString(),

courier.ReceiveInteger());

break;

case ClientRequests.CloseConnection:

default:

ViewInfoLeave();

courier.Leave();

return;

}

}

catch (SocketException e)

{

Console.WriteLine("Problem: \n" + e);

return;

}

}

}

private void SendOpenFolderBrowser()

{

string[] drives = FileManager.GetDrives();

courier.SendInteger(drives.Length);

foreach (string drive in drives)

{

courier.SendString(drive);

}

}

private void SendFiles(string path, bool hidden)

{

path = PathUtility.ClosePath(path);

string[] files = hidden ? FileManager.GetFiles(path)

: FileManager.GetFilesWithoutHidden(path);

courier.SendInteger(files.Length);

foreach (string file in files)

{

courier.SendString(file);

}

courier.SendString(FileManager.FormatSize(path, 2));

}

private void SendAttributes(string file)

{

courier.SendInteger((int)File.GetAttributes(file));

}

private void SetAttributes(string file, int attributes)

{

File.SetAttributes(file, (FileAttributes)attributes);

}

private void SendSubFolders(string path, bool hidden)

{

path = PathUtility.ClosePath(path);

string[] directories = hidden ? FileManager.GetDirectories(path)

: FileManager.GetDirectoriesWithoutHidden(path);

courier.SendInteger(directories.Length);

foreach (string directory in directories)

{

courier.SendString(PathUtility.GetNameWithoutParent(directory));

}

}

private ClientRequests WaitRequest()

{

return (ClientRequests) courier.ReceiveInteger();

}

private void ViewInfoLeave()

{

Console.WriteLine("Client leave: {0}", courier.GetSocket().RemoteEndPoint);

Console.WriteLine("[TIME] {0}\n", DateTime.Now);

}

}

}

FileManager.cs

using System;

using System.Collections.Generic;

using System.IO;

namespace MyAddedTypes

{

class FileManager

{

public static string[] GetFilesWithoutHidden(string path)

{

return PathsWithout(GetFiles(path),

FileAttributes.Hidden);

}

public static string[] GetDirectoriesWithoutHidden(string path)

{

return PathsWithout(GetDirectories(path),

FileAttributes.Hidden);

}

public static string[] GetFiles(string path)

{

string[] paths;

try

{

paths = Directory.GetFiles(path);

}

catch (UnauthorizedAccessException)

{

paths = new string[0];

}

return PathsWithout(paths, FileAttributes.System);

}

public static string[] GetDirectories(string path)

{

string[] paths;

try

{

paths = Directory.GetDirectories(path);

}

catch (UnauthorizedAccessException)

{

paths = new string[0];

}

return PathsWithout(paths, FileAttributes.System);

}

public static string[] GetDrives()

{

DriveInfo[] drives = DriveInfo.GetDrives();

List<string> listDrives = new List<string>();

foreach (DriveInfo drive in drives)

{

try

{

Directory.GetFiles(drive.ToString());

listDrives.Add(drive.ToString());

}

catch (IOException) { }

catch (UnauthorizedAccessException) { }

}

return listDrives.ToArray();

}

public static string FormatSize(string path, int nums)

{

int uSize = 0;

double fSize = DirSize(path);

string[] unit = { "b", "kB", "MB", "GB", "TB" };

while (fSize >= 1024 && uSize < unit.Length)

{

fSize /= 1024;

uSize++;

}

return string.Format("{0} {1}",

Math.Round(fSize, nums), unit[uSize]);

}

private static string[] PathsWithout(string[] paths,

FileAttributes fileAttributes)

{

List<string> listWithoutSystem = new List<string>();

foreach (string path in paths)

{

if (!File.GetAttributes(path).HasFlag(fileAttributes))

{

listWithoutSystem.Add(path);

}

}

return listWithoutSystem.ToArray();

}

private static long DirSize(string path)

{

long size = 0;

string[] files = GetFiles(path);

foreach (string file in files)

{

size += (new FileInfo(file)).Length;

}

string[] directories = GetDirectories(path);

foreach (string directory in directories)

{

size += DirSize(directory);

}

return size;

}

}

}

ThreadMonitor.cs

using System.Collections.Generic;

using System.Threading;

namespace MyAddedTypes

{

class ThreadMonitor

{

private List<Thread> threads;

public ThreadMonitor(List<Thread> threads)

{

this.threads = threads;

}

public void MonitorThreads()

{

while (true)

{

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

{

if (!threads[i].IsAlive)

{

threads[i].Abort();

threads.RemoveAt(i);

}

}

}

}

}

}

Courier.cs

using System;

using System.Text;

using System.Net.Sockets;

namespace MyAddedTypes

{

class Courier

{

private Socket socket;

public Courier(Socket socket)

{

this.socket = socket;

}

public void SendInteger(int number)

{

byte[] sendBytes = BitConverter.GetBytes(number);

socket.Send(sendBytes);

}

public void SendString(string line)

{

byte[] sendBytes = Encoding.ASCII.GetBytes(line);

SendInteger(sendBytes.Length);

socket.Send(sendBytes);

}

public string ReceiveString()

{

int size = ReceiveInteger();

byte[] receiveBytes = new byte[size];

int lengthBytes = socket.Receive(receiveBytes);

return Encoding.ASCII.GetString(receiveBytes, 0, size);

}

public int ReceiveInteger()

{

byte[] receiveBytes = new byte[sizeof(Int32)];

socket.Receive(receiveBytes);

return BitConverter.ToInt32(receiveBytes, 0);

}

public Socket GetSocket()

{

return socket;

}

public void Leave()

{

socket.Shutdown(SocketShutdown.Both);

socket.Close();

}

}

}

PathUtility.cs

using System.Collections.Generic;

namespace MyAddedTypes

{

class PathUtility

{

public static string AddToPath(string path, string name)

{

path = ClosePath(path);

path += name;

return ClosePath(path);

}

public static string DecrementPath(string path)

{

if (path.IndexOf('\\') != path.LastIndexOf('\\'))

{

string temp = path.Remove(path.Length - 1);

path = path.Remove(temp.LastIndexOf('\\') + 1);

}

return path;

}

public static string GetNameWithoutParent(string path)

{

if (path.LastIndexOf('\\') > -1)

{

return path.Remove(0, path.LastIndexOf('\\') + 1);

}

return path;

}

public static string FindFullPath(string name,

List<string> pathList)

{

foreach (string path in pathList)

{

if (path.IndexOf(name) > -1) return path;

}

return name;

}

public static string ClosePath(string path)

{

if (path != "" && path[path.Length - 1] != '\\')

return path + '\\';

return path;

}

}

}

ClientRequests.cs

namespace MyAddedTypes

{

enum ClientRequests

{

OpenFolderBrowser,

ShowSubFolders,

SelectFolder,

GetAttributes,

SetAttributes,

CloseConnection

}

}